НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки
Автоматизованих систем обробки інформації і управління

«На правах рукопису»
УДК _____

«До захисту допущено»
В.о. завідувача кафедри
_____ О.А.Павлов
«___»_____20__ р.

Магістерська дисертація

на здобуття ступеня магістра
зі спеціальності 121 Інженерія програмного забезпечення
на тему: «Математичне та програмне забезпечення для побудови навігаційних маршрутів»

Виконав (-ла):
студент (-ка) VI курсу, групи ІП-71мн
Храмченко Микола Сергійович _____
Керівник:
доц., доц., к.т.н. Муха І.П. _____
Консультант:
доц., к.т.н. Ліщук К.І. _____

Рецензент:
_____ _____

Засвідчую, що у цій магістерській дисертації немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2019 року

Національний технічний університет України
«Київський політехнічний інститут імені Ігоря Сікорського»
Факультет інформатики та обчислювальної техніки
Автоматизованих систем обробки інформації і управління

Рівень вищої освіти – другий (магістерський)

Спеціальність – 121 «Інженерія програмного забезпечення

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ О.А.Павлов
«___»_____20__ р.

ЗАВДАННЯ
на магістерську дисертацію студенту
Храмченко Микола сергійович

1. Тема дисертації «**Математичне та програмне забезпечення для побудови навігаційних маршрутів**», науковий керівник дисертаціїМуха Ірина Павлівна, затверджені наказом по університету від «___»_____ 2019 р. №_____

2. Термін подання студентом дисертації_____

3. Об’єкт дослідження: Процес побудови навігаційних маршрутів, з урахуванням критеріїв оптимальності

4. Предмет дослідження: Методи та алгоритми, які використовуються для оптимізації маршрутів згідно множини критеріїв одночасно

5. Перелік завдань, які потрібно розробити: аналіз існуючих методів розпізнавання побудови навігаційних маршрутів, розробка методів побудови багатокритеріальних навігаційних маршрутів; дослідження ефективності розробленого алгоритму.

6. Орієнтовний перелік графічного (ілюстративного) матеріалу: _____

_____

7. Орієнтовний перелік публікацій 2 публікації

8. Консультанти розділів дисертації

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Графічний	Ліщук К.І., доц.		

9. Дата видачі завдання _____

Календарний план

№ з/п	Назва етапів виконання магістерської дисертації	Термін виконання етапів магістерської дисертації	Примітка
1	Аналіз існуючих методів побудови навігаційних маршрутів		
2	Аналіз методів багатокритеріальної оптимізації		
3	Розробка алгоритму побудови навігаційних маршрутів з урахуванням додаткових критеріїв.		
4	Дослідження ефективності розробленого алгоритму.		
5	Розробка технічної документації.		

Студент Храмченко М.С.

Науковий керівник дисертації Муха І.П.

РЕФЕРАТ

Магістерська дисертація: 122 с., 36 рис., 8 табл., 4 додатка, 20 джерел.

Актуальність теми.

Рівень розвитку сучасних технологій дозволяє обладнувати мобільні пристрої (мобільні телефони, кишенькові та планшетні комп'ютери) засобами доступу до бездротового мобільного Інтернету та до глобальної системи супутникової навігації GPS. Це дає змогу організувати наземну навігацію рухомих об'єктів, зокрема, автомобільну, використовуючи мобільні навігаційні додатки.

Однією із найбільш важливих задач таких навігаторів є побудова оптимального маршруту руху об'єкта до заданої точки.

Наявні способи реалізації даного функціоналу в існуючих програмних засобах (таких як GoogleMaps, YandexMaps, AppleMaps та інші) передбачають при побудові маршруту можливість обрати на власний розсуд, як критерій оптимальності, найкоротшу відстань або найменший час. За замовчуванням система обирає найшвидший маршрут. Якщо існує два маршрути з однаковим часом проходження, система автоматично обирає найкоротший із них та пропонує альтернативний варіант.

Значним недоліком таких мобільних навігаторів є обмеженість вибору критеріїв оптимізації при побудові маршруту. Наприклад, відсутня можливість вибору маршруту з урахуванням якості дорожнього покриття, завантаженості доріг, якості їх освітленості, інформації про аварійні ситуації, ремонтні роботи, пробки, закриття дороги або зміну напрямку руху, допустимого відхилення від найменшого часу або найкоротшої дистанції тощо.

Тому мобільні навігатори, що забезпечують побудову навігаційних маршрутів з можливістю урахування додаткових критеріїв (таких як якість

дорожнього покриття, освітленість дороги тощо), є дуже затребуваними, а їх розробка - вкрай актуальним завданням.

Мета і задачі дослідження.

Мета даної роботи полягає в дослідженні та розробці математичних та програмних засобів для багатокритеріальної побудови навігаційних маршрутів.

Для досягнення поставленої мети необхідно розв‘язати комплекс наступних взаємопов‘язаних *задач*:

– дослідити існуючі методи та підходи до побудови навігаційних маршрутів;

– дослідити існуючі методи багатокритеріальної оптимізації;

– розробити математичне забезпечення для побудови багатокритеріальних навігаційних маршрутів;

– розробити мобільний навігатор, який має підтримувати функціонал побудови навігаційних маршрутів з урахуванням декількох критеріїв оптимальності;

– виконати експериментальне дослідження запропонованих рішень.

Об’єкт дослідження.

Процес побудови навігаційних маршрутів.

Предмет дослідження.

Методи багатокритеріальної оптимізації та алгоритми знаходження оптимальних шляхів у графі.

Наукова новизна одержаних результатів полягає у розробці метода побудови багатокритеріальних навігаційних маршрутів, який ґрунтується на використанні удосконаленого алгоритма A*.

Практичне значення одержаних результатів.

Всі запропоновані математичні методи і алгоритми доведені до практичної реалізації у рамках програмного забезпечення, яке

використовується для побудови багатокритеріальних навігаційних маршрутів з урахуванням відстані і якості дорожнього покриття.

Публікації.

Матеріали роботи опубліковані в тезах конференції «Інформатика та обчислювальна техніка-ІОТ-2017» та «Інформаційні системи та технології управління - 2019»

Ключові слова.

БАГАТОКРИТЕРІАЛЬНА ОПТИМІЗАЦІЯ, МЕТОД ЗГОРТАННЯ КРИТЕРІЇВ, АЛОГОРИТМИ ПОШУКУ МАРШРУТИВ, ЗНАХОДЖЕННЯ ОПТИМАЛЬНОГО ШЛЯХУ, А*, ПОБУДОВА БАГАТОКРИТЕРІАЛЬНИХ МАРШРУТІВ.

ABSTRACT

Master's dissertation: 122 pages, 36 images, 8 tables, 4 annexes, 20 sources.

The level of development of modern technologies allows us to equip mobile devices (mobile phones, pocket and tablet computers) with means of access to wireless mobile Internet and to the global system of satellite navigation GPS. This allows you to organize land-based navigation of moving objects, in particular, automotive, using mobile navigation applications.

One of the most important tasks of such navigators is the construction of the optimal route of the object to a given point.

Available ways to implement this functionality in existing software tools (such as GoogleMaps, YandexMaps, AppleMaps, etc.) provide the option to choose at your own discretion as a criterion of optimality, the shortest distance, or the smallest time. By default, the system chooses the fastest route. If there are two routes with the same transit time, the system automatically chooses the shortest of them and offers an alternative.

A significant disadvantage of such mobile browsers is the limited choice of optimization criteria when constructing a route. For example, there is no possibility to choose the route taking into account the quality of the road surface, the traffic load, the quality of its illumination, emergency information, repair work, traffic jam, road closure or change of direction of travel, permissible deviation from the shortest time or the shortest distance, etc.

Therefore, mobile navigators, which provide the construction of navigation routes with the possibility of taking into account additional criteria (such as road surface quality, illumination of the road, etc.), are very in demand, and their development is a very topical task.

The main purpose of this work is to research and develop mathematical and software tools for multicriteria construction of navigation routes.

To achieve this goal, it is necessary to solve a set of the following interconnected tasks:

– to explore existing methods and approaches to constructing navigation routes;

– to explore existing multi-criteria optimization methods;

– develop mathematical support for construction of multi-criteria navigation routes;

– to develop a mobile navigator that should support the function of constructing navigation routes, taking into account several criteria of optimality;

– carry out an experimental study of the proposed solutions.

Object of study.

The process of constructing navigation routes.

Subject of study.

Methods of multi-criteria optimization and algorithms for finding optimal paths in a graph.

The scientific novelty of the results obtained is the development of a method for constructing multicriteria navigation routes, which is based on the use of the advanced algorithm A *.

The practical value of the results obtained.

All proposed mathematical methods and algorithms are brought to practical realization within the framework of the software used to construct multicriteria navigational routes taking into account the distance and quality of the road surface.

Publications

The materials of the work are published in the theses of the conference "Informatics and Computing Technology-IOT-2017" and "Information Systems and Technologies of Management - 2019"

Keywords.

MULTI-CRITERIAN OPTIMIZATION, METHOD OF CRITERIA BUILDING, ALGORITHM OF ROUTINE SEARCH, INTRODUCTION OF OPTIMAL WAY, A *, BUILDING OF MULTI-CRITERIAN ROUTES.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, СКОРОЧЕНЬ І ТЕРМІНІВ

GPS – Global Position System

OSM – Open Street Maps

API – Application Programming Interface

MVC – Model View Controller

WWDC – World Wide Developer Conference

RIP – Routing Information Protocol

XML – eXtensible Markup Language

LLVM - Low Level Virtual Machine

ВСТУП

Рівень розвитку сучасних технологій дозволяє обладнувати мобільні пристрої (мобільні телефони, кишенькові та планшетні комп’ютери) засобами доступу до бездротового мобільного Інтернету та доступу до глобальної системи навігації GPS. Це дає змогу організувати навігацію, що значно прискорює процес пошуку об’єктів на незнайомій місцевості, як приклад - пошук організації за адресою.

GPS - система глобального позиціонування (англ.*Global Positioning System*) — сукупність радіоелектронних засобів, що дозволяє визначати положення та швидкість руху об'єкта на поверхні Землі або в атмосфері. Положення об'єкту обчислюється завдяки використанню GPS-приймача, який приймає та обробляє сигнали супутників GPS. GPS система в своєму складі має наземні центри управління [1].

З появою GPS з'явилася можливість використовувати додатки, які досить точно показують розташування користувача на місцевості. Програми, що використовують GPS, існують для будь-яких сучасних мобільних платформ і вони досить гідно увійшли до нашого життя в усі сфери людської діяльності. Зараз складно уявити собі який-небудь транспорт без навігаційної системи, нехай то буде літак, автомобіль або судно. GPS використовують будівельники, військові, геологи і цей список можна продовжувати довго.

Варто відмітити, що всі навігаційні системи можна умовно розділити на два типи:

– промислові - великі системи, призначені вирішувати логістичні задачі, наприклад, система управління аеропортом, яка стежить за становищем кожного літака [1];

– персональні - допомагають користувачеві не загубитися на невідомій місцевості і прокласти маршрут до пункту призначення, наприклад, автомобільний навігатор [1].

Сучасні персональні навігаційні системи при побудові маршруту в основному обирають найкоротший або найшвидший маршрут, що не завжди є бажаним. Можна навести безліч прикладів, коли вкрай необхідна невелика проста система для побудови навігаційних маршрутів, основним завданням якої була б побудова багатокритеріальних маршрутів. Наприклад, така система знадобилася б людям, що прямують від одного міста в інше та бажають обрати дорогу з якіснішим покриттям. Також така система могла знадобитися людям що подорожують в нічний час та хочуть прямувати по освітленим ділянкам дороги. У великому місті автомобілісти можуть побудувати маршрут в об’їзд пробок.

Таким чином, засоби, що забезпечують побудову навігаційних маршрутів з можливістю урахування додаткових критеріїв (таких як якість дорожнього покриття, освітленість, тощо), є дуже затребуваними, а їх розробка і впровадження вкрай актуальним завданням.

Основна мета даної роботи полягає в дослідженні та розробці математичних та програмних засобів для багатокритеріальної побудови навігаційних маршрутів.

Для досягнення поставленої мети необхідно розв‘язати комплекс наступних взаємопов‘язаних задач:

– дослідити існуючі методи та підходи до побудови навігаційних маршрутів;

– дослідити існуючі методи багатокритеріальної оптимізації;

– розробити математичне забезпечення для побудови багатокритеріальних навігаційних маршрутів;

– розробити мобільний навігатор, який має підтримувати функціонал

побудови навігаційних маршрутів з урахуванням декількох критеріїв оптимальності;

- виконати експериментальне дослідження запропонованих рішень.

1 ОГЛЯД ПІДХОДІВ ДО ПОБУДОВИ НАВІГАЦІЙНИХ МАРШРУТІВ

1.1 Загальні положення

Сучасні навігаційні системи мають багато різних варіантів реалізації, починаючи з онлайнових сервісів, доступних на будь-якому комп'ютері, і закінчуючи мобільними пристроями, які завжди можна взяти з собою.

Задача побудови оптимального маршруту, вперше піднята в 1832 році в книзі «Комівояжер - як він повинен вести себе і що повинен робити для того, щоб доставляти товар і мати успіх у своїх справах - поради старого кур'єра », є актуальною і на сьогоднішній день - розробляються нові методи розв'язання задачі, реалізуються програми, які дозволяють працювати з кількістю вузлів близьким до мільйону за прийнятний час. Незгасний інтерес до цієї задачі обумовлений різноманітністю застосувань її на практиці. Пошук оптимального шляху широко використовується у всіх задачах транспортної логістики [2].

Задача Комівояжера була вперше розглянута з математичної точки зору в 1930-их роках математиком і економістом Карлом Менеджером. У 1940-их статистики Механобіс, Джессен, Гош і Маркс намагалися знайти застосування задачі комівояжера в сільськогосподарському секторі, що призвело до її популяризації - починаючи з середини 1950х років методи розв'язання задачі стали публікуватися в наукових журналах [2].

1.2 Огляд існуючих навігаційних додатків

1.2.1 Автомобільні навігаційні пристрої

Автомобільні навігаційні системи зараз дуже поширені, мають різні форм-фактори, проте завжди складаються з:

– графічного екрану, на якому відображається карта, місце перебування і прокладений маршрут;

– системи введення натисканням на екран або винесеними на корпус клавішами;

– динаміка, здатного відтворювати в достатній для сприйняття якості людський голос, який озвучує інструкції до переміщення.

Навігатори компанії Navitel (рисунок 1.1) мають безліч різних реалізацій, в тому числі для різних платформ. При цьому всі вони схожі по функціоналу і зовнішньому вигляду інтерфейса користувача. Система реалізує два варіанти опису маршруту: зображення траєкторії і покрокова інструкція до переміщення. Доступний функціонал для завдання стартовою і фінішної точок маршруту, а також уточнення його додатковими транзитними точками, що за своєю сутністю розбиває маршрут на кілька послідовних шляхів. Маршрут між точками обирається в залежності від одного критерію – відстані [3].

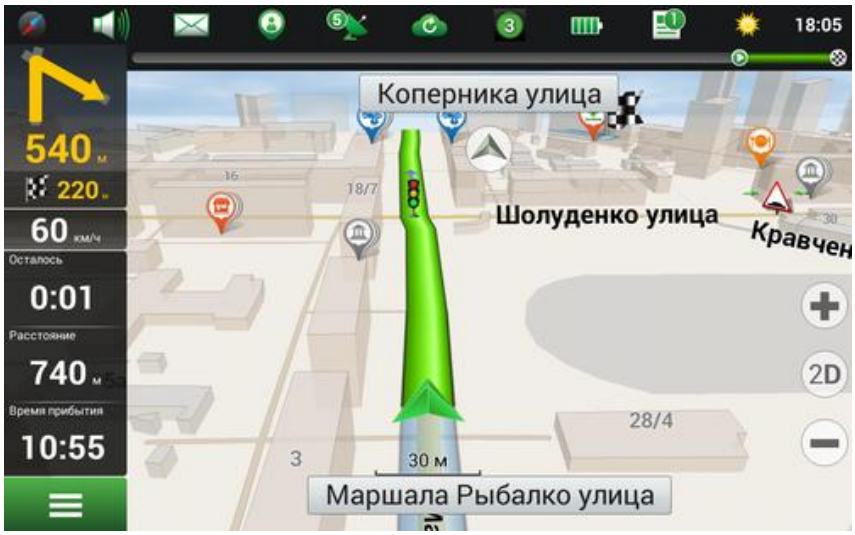

Рисунок 1.1 – Навігатор Navitel

Навігатори компанії Garmin (рисунок 1.2) також налічують велику кількість варіантів реалізацій і форм-факторів, при цьому залишаючись стабільними у плані пропонованого користувачам функціоналу і інтерфейсу користувача. Система аналогічно реалізує два варіанти опису маршруту:

зображення траєкторії і покрокова інструкція переміщення. Введення опису маршруту, як і у випадку з Navitel, може бути здійснене тільки шляхом завдання стартової і кінцевої точки з можливістю додавання транзитних місць. Подальше визначення траєкторії шляху відбувається в автоматичному режимі.Критерій вибору траєкторії руху – найкоротша відстань між початковою та кінцевою точками руху [4].

Рисунок 1.2 – Навішатор Garmin

Навігатори компанії Mio (рисунок 1.3) давно представлені на ринку. У них аналогічно існує велика кількість варіантів реалізацій зовнішнього вигляду, але при цьому навігатори залишаються постійними в плані інструментів, що вони пропонують. Реалізовано два варіанти опису маршруту: зображення траєкторії і покрокова інструкція до переміщення. Введення опису маршруту нічим не відрізняється від інших систем і відбувається через завдання контрольних і опорних точок. Критерієм оптимальності маршруту як і в попередніх випадках являється найкоротша відстань.

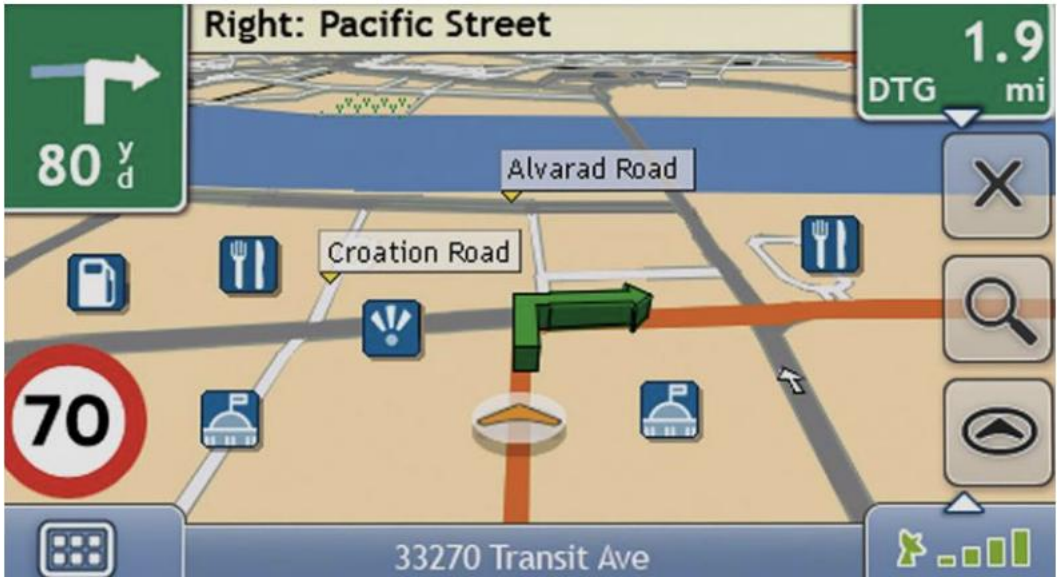

Рисунок 1.3 – Навігатор Mio

Спільною рисою для більшості (але не для всіх) автомобільних навігаторів є робота в автономному режимі, що означає:

– інформація про завантаженість доріг недоступна;

– інформація про дороги та знаки оновлюється рідко і вимагає під'єднання до комп'ютера;

– персональна інформація про персональні маршрути, що часто використовуються, є локальною і не синхронізується з іншими пристроями користувача;

– навігатори автоматично не підлаштовують свої алгоритми під користувача на основі історії його переміщень і не пропонують засобів спрощення опису маршрутів на основі відомих йому ділянок шляху;

– додавання критеріїв (наприклад якість дорожнього покриття) для побудови маршрутів є неможливим. Єдиним критерієм оптимальності в даних рішеннях вважається найкоротша відстань.

1.2.2 Мобільні навігаційні додатки

Смартфони і планшети можуть вирішувати ті ж самі завдання, що й автомобільні навігатори. Для цього необхідне встановлення спеціального навігаційного програмного додатка. Розглянемо наступні програмні рішення:

– Apple Maps;

– Google Maps;

– Яндекс Карти.

Всі розглянуті в даному розділі навігаційні системи являють собою клієнтські програми зі зберіганням інформації на віддалених серверах. Таким чином, програмна система ділиться на дві частини:

Серверна частина:

– зберігає інформацію про карти, маршрути та інше;

– прокладає маршрути;

– доступна по мережі Інтернет з будь-якої точки світу;

– має картографічну інформацію, що постійно оновлюється і актуалізується відповідно зі змінами в реальному світі.

Клієнтська частина:

– зберігає локальні налаштування користувача;

– кеширує останню отриману від сервера інформацію на деякий час, визначений розробником, після чого перевіряє, чи є в ній зміни і в разі їх наявності запитує у сервера оновлені дані;

– візуалізує отримані від сервера дані в зручному для користувача вигляді;

– відслідковує положення користувача і відображає його поточні координати.

Мобільна навігаційно-картографічна система Apple Maps від однойменної компанії Apple працює тільки на пристроях компанії, пропонує

вже стандартні способи опису маршруту у вигляді зображення траєкторії (рисунок 1.4) і покрокової інструкції до переміщення (рисунок 1.5), як це показано на ілюстраціях нижче.

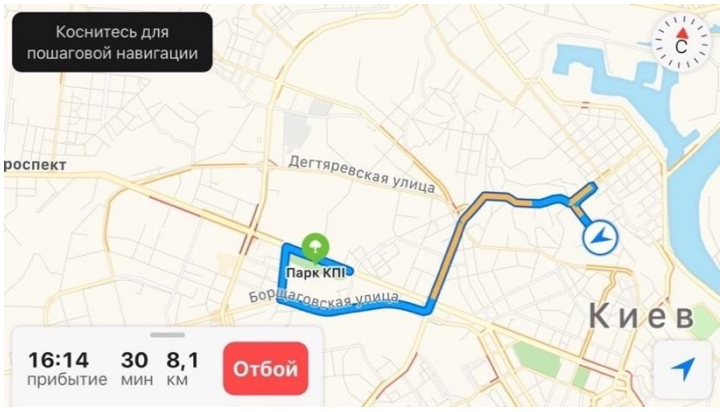

Рисунок 1.4 – Мобільний додаток Apple Maps

Введення конкретного шляху в системі Apple Maps неможливе, є тільки функціонал зі завданням стартовою і кінцевої точки маршруту, далі він розраховується автоматично на серверній стороні додатку. Додавання проміжних точок трансферу, як в автомобільних навігаторах, також неможливо. Як основний критерій вибору оптимального маршруту вважається найменший час слідування за маршрутом. В останніх версіях програмного засобу система враховує завантаженість доріг, що прямопропорційно впливає на час руху. Система сама обирає маршрут, без надання користувачу права вибору.

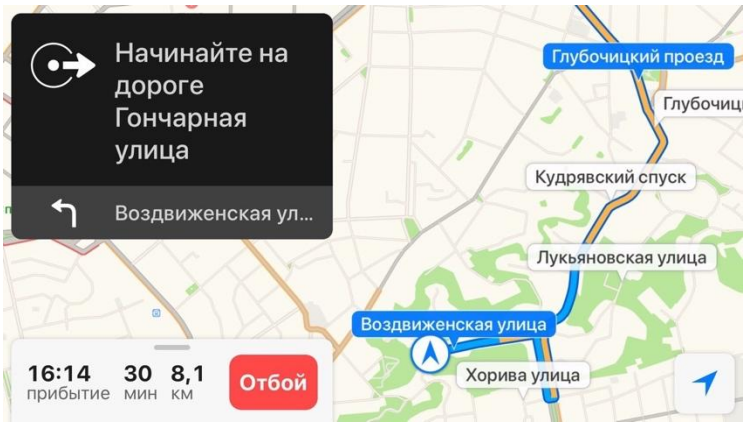

Рисунок 1.5 – Покрокова інструкція до переміщення

Навігаційна система Google Maps від компанії Google реалізована для безлічі платформ, у тому числі для пристроїв Apple і пристроїв на базі власної операційної системи Android. Функціонал мало відрізняється від розглянутої вище системи Apple Maps. Ті ж самі способи опису маршруту (рисунок 1.6): візуальне уявлення траєкторії шляху і покрокова інструкція до переміщення без урахування персональних знань користувача про оточуючі об'єкти і історії його переміщень в минулому. Програмний засіб надає користувачу право обрати критерій оптимальності маршруту: найкоротша відстань, найменший час, найпопулярніший серед користувачів. За замовчуванням система обирає найшвидший маршрут. Якщо існує два схожих за часом маршрута система автоматично обирає найкоротший та пропонує альтернативний варіант.

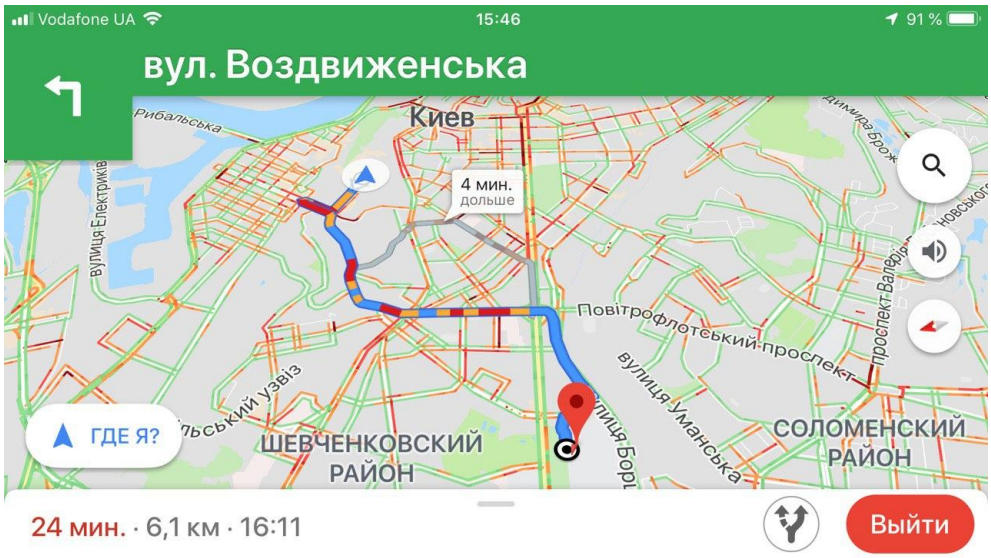

Рисунок 1.6 – Мобільний додаток Google Maps

Система Яндекс Карти від компанії Яндекс широко поширена в основному на території СНД (На сьогоднішній день заблокована указом президента на території України). Додаток сконцентровано на надання довідкової інформації по розташуванню об'єктів і поточному стану пробок. Функціонал по формуванню опису прокладеного маршруту можливий тільки

у вигляді візуального представлення траєкторії (рисунок 1.7). Введення маршруту можливе тільки через початкову та кінцеву точку без функціоналу щодо його уточнення. Так як і в Google Maps система орієнтується на два основні критерії: найкоротша відстань та найменший час. Доступна інформація про завантаженість доріг та ДТП. Це допомагає точніше підрахувати час слідування і у випадку великої затримки запропонувати альтернативний маршрут.

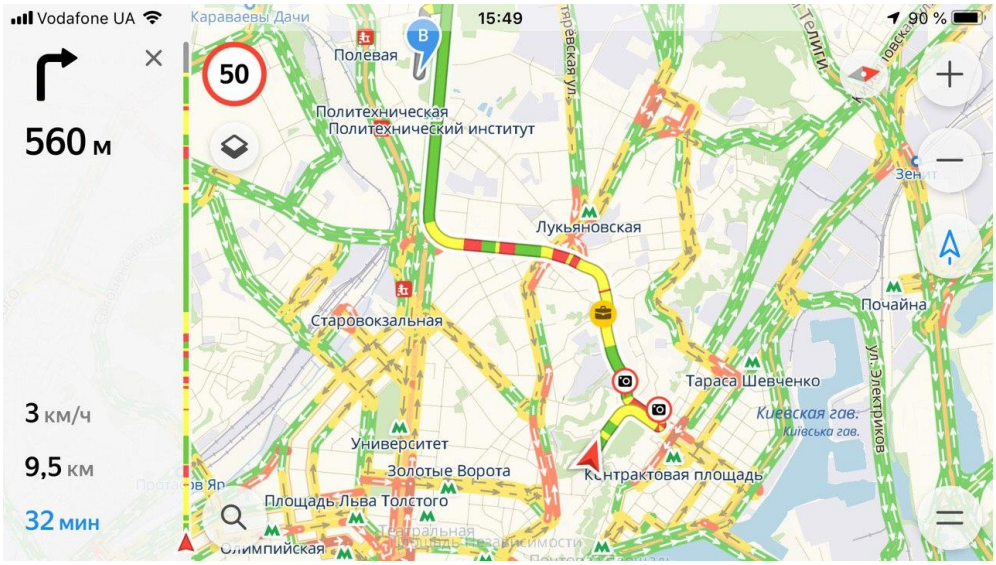

Рисунок 1.7 – Мобільний додаток Yandex Maps

1.2.3 Переносні навігаційні пристрої

Смарт-годинник і смарт-окуляри почали виготовлятися порівняно недавно, але вже на поточний момент можуть допомагати користувачеві в навігації. Програмні рішення знаходяться в стадії глибокої розробки. Їх розгляд на поточний момент не відображає підсумкового якості продукту. Порівнянню підлягалися платформи:

- Google Wear;
- Apple Watch ;
- Google Glass.

Комп'ютерна система Google Wear (рисунок 1.8) виробництва компанії Google працює у зв'язці зі смартфоном, передаючи системні дані по каналу Bluetooth. Навігаційний додаток на смарт-годиннику під'єднується до Google Maps і здійснює свого роду вивід на додатковий екран, доступний на зап'ясті користувача, коли сам смартфон знаходиться в кишені або сумці. При запуску навігації до точки, заданої за допомогою голосової команди з використанням системи розпізнавання мови Google Now, на екрані смарт-годинника відображається повністю весь шлях у вигляді зображення траєкторії і потребує підтвердження користувачем на початок навігації за даним маршрутом. Маршрут при цьому прокладається автоматично від поточного місця розташування. Маршрути будуються на тих же серверах що і для мобільного додатку, а отже і критерії враховуються такі ж.

Рисунок 1.8 – Комп'ютерна система Google Wear

Смарт-годинник Apple Watch (рисунок 1.9) виробництва компанії Apple пропонує аналогічний функціонал. За початкову точку шляху приймається поточне місце розташування користувача, кінцева точка задається через голосового асистента Siri. Маршрут прокладається в автоматичному режимі без можливості його зміни. Перед початком навігації

користувачеві показується шлях цілком у вигляді зображення траєкторії на карті місцевості. Критерій оптимальності – відстань.

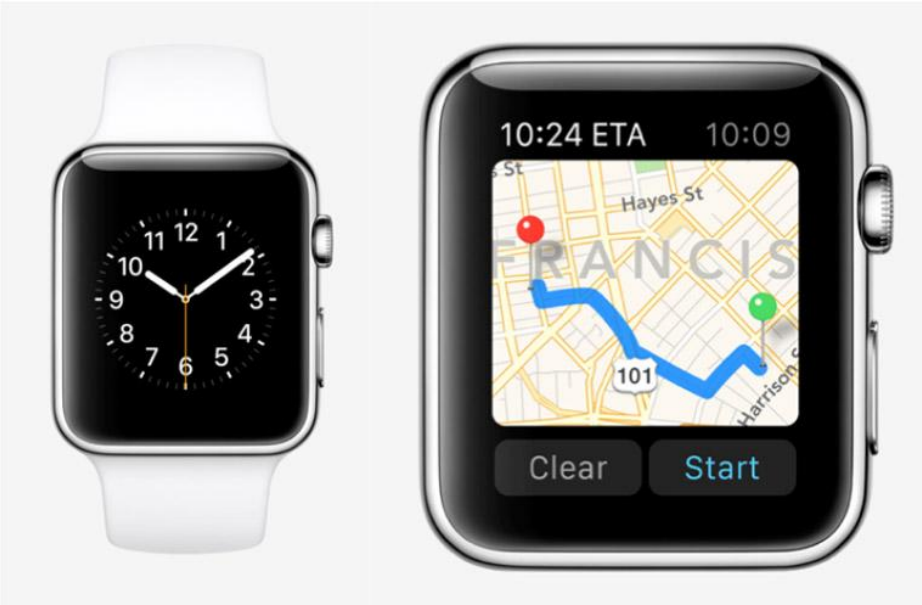

Рисунок 1.9 - Смарт-годинник Apple Watch

Аналогічним в плані навігаційно-картографічного функціоналу є смарт-окуляри Google Glass (рисунок 1.10) від Google. Пристрій закріплюється на голові користувача, як звичайні окуляри, з правого боку розташований маленький проектор, що проектує зображення на внутрішню сторону правого ока таким чином, що користувач нібито бачить голограму, що висить у повітрі на деякій відстані від особи. Додаток дуже схожий на згадані вище додатки для смарт-годинників. Тим же способом, що і в Google Wear, користувач задає кінцеву точку маршруту за допомогою голосового помічника Google Now, за стартову точку береться поточне місце розташування, передане на окуляри зі смартфона, до якого вони приєднані по бездротовому каналу Bluetooth. Перед початком навігації на екрані відображається зображення траєкторії всього маршруту.

Рисунок 1.10 - Google Glass

1.2.4 Онлайнові навігаційні системи

Крім розглянутих вище окремих пристроїв і програм для виконання навігаційніх завдань існують онлайнові системи, доступні безпосередньо з веб-браузерів та не потребують ніяких встановлень локально. Подібні системи працюють переважно на стаціонарних комп'ютерах, але можуть бути також запущені на смартфонах і планшетах (даний варіант використання є порівняно рідкісним, бо для мобільних пристроїв існують спеціалізовані навігаційні програми, розглянуті вище). Розглянемо наступні онлайнові навігаційні системи:

- Google Maps (рисунок 1.11);
- Яндекс Карти (рисунок 1.12);
- Bing Maps (рисунок 1.13).

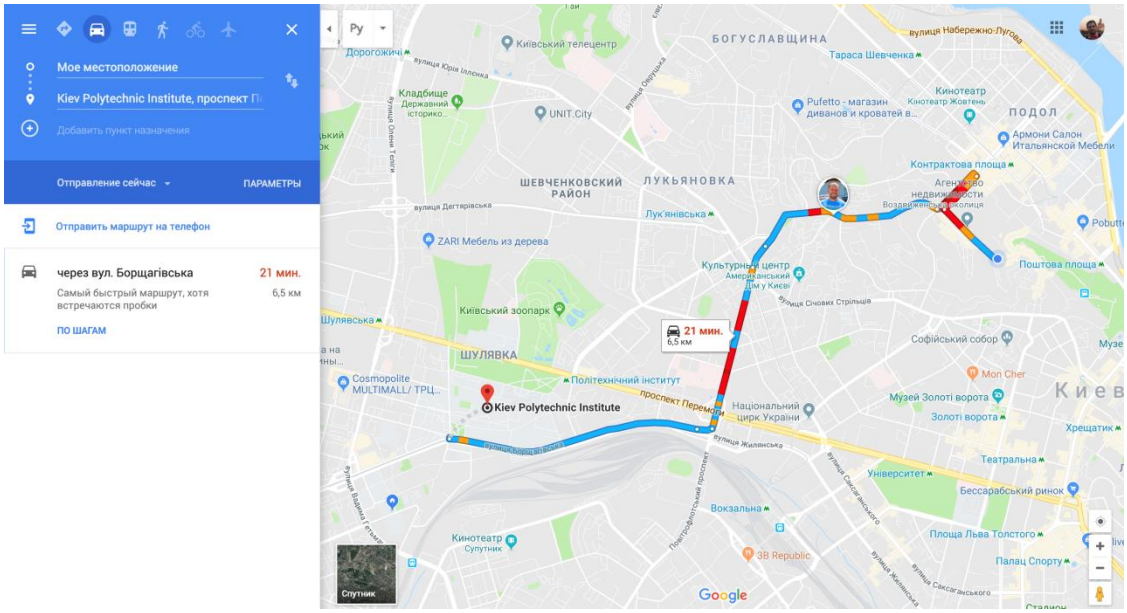

Рисунок 1.11 – Web-версія Google Maps

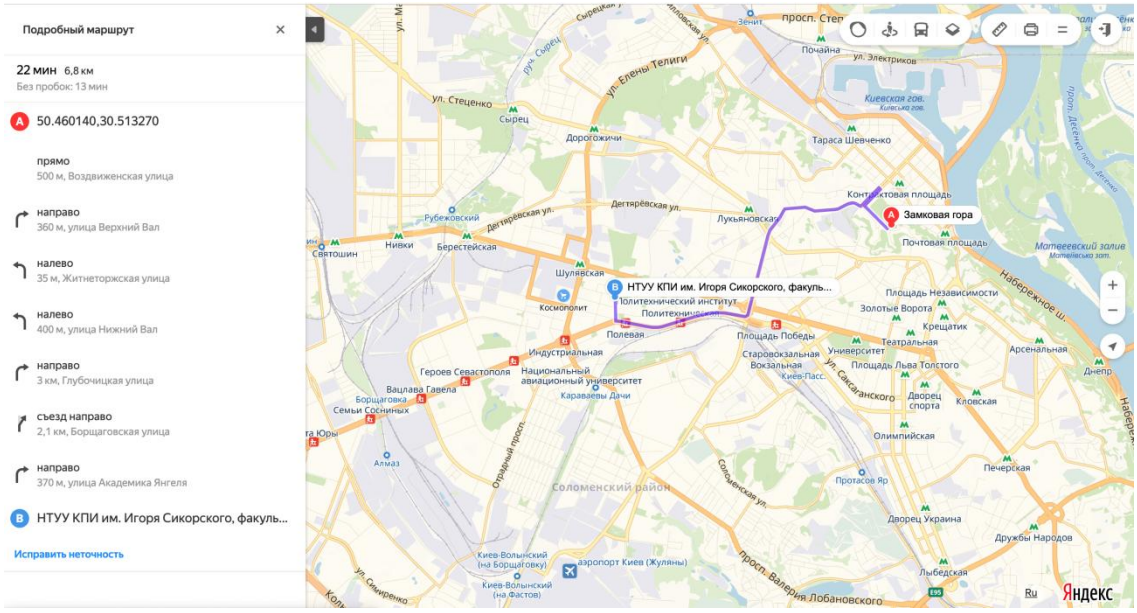

Рисунок 1.12 - Web-версія Yandex Maps

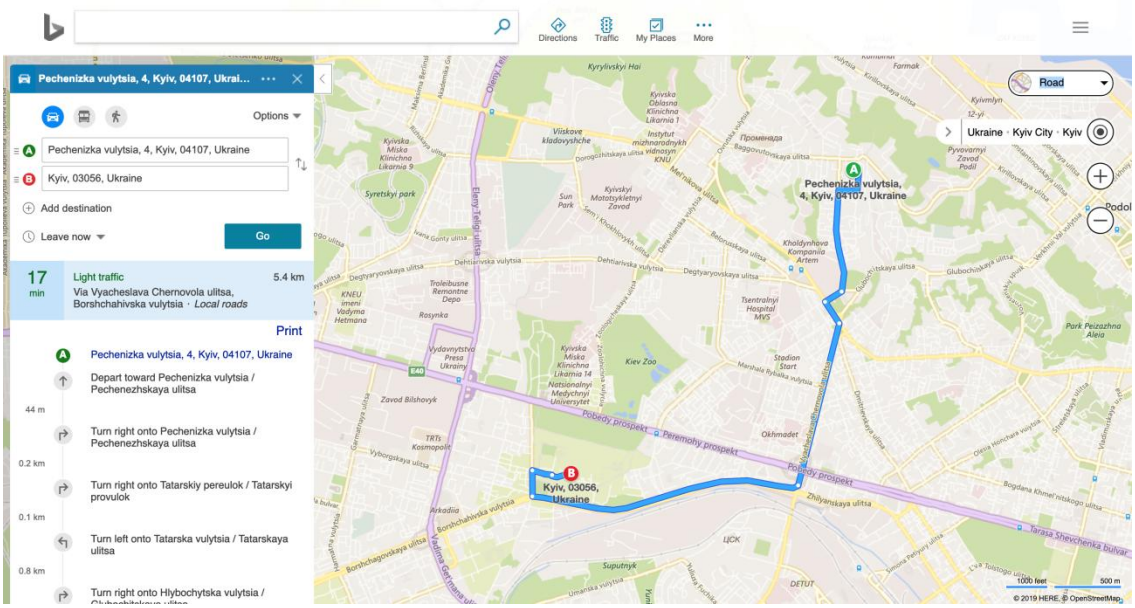

Рисунок 1.13 - Web-версія Bing Maps

Всі три розглянуті онлайнові системи пропонують абсолютно ідентичний функціонал і відрізняються переважно:

– зовнішнім виглядом і зручністю призначеного для користувача інтерфейсу;

– картографічної базою і списком відомих системі місць, по яким можна здійснювати пошук.

Онлайнові системи пропонують опис маршруту в двох існуючих варіантах: зображення траєкторії і докладна покрокова інструкція до переміщення. Введення маршруту можливий двома способами:

– завдання початкової і кінцевої точки (в якості початкової точки може виступати, в тому числі, поточне місце розташування користувача, яке визначається або по GPS, або за даними від провайдера мобільного інтернету, або по геолокаційним даними Wi-Fi точки доступу) і автоматичне прокладання між ними маршруту з урахуванням критеріїв: відстань, час, завантаженість доріг;

− завдання початкової і кінцевої точок і зміна прокладеного маршруту за допомогою перетягування опорних точок мишкою по карті так, що в результаті можна домогтися бажаного маршруту (такий функціонал доступний лише на стаціонарних комп'ютерах і ноутбуках, що використовують в якості маніпулятора комп'ютерну мишу).

Варто зазначити, що всі перераховані навігаційні системи враховують правила дорожнього руху та будують маршрут згідно з ними.

1.3 Огляд існуючих алгоритмів пошуку оптимального шляху

Алгоритми для вирішення задачі комівояжера можна розділити на точні (exact algorithm) і неточні (non-exact algorithm). Точні алгоритми включають в себе перебір всіх можливих варіантів, в окремих випадках рішення можуть бути швидко знайдені, але в цілому здійснюється перебір n! циклів. Другі в загальних випадках застосовуються для задач, які неможливо вирішити точно (обчислення визначених інтегралів, рішення нелінійних рівнянь, підрахунок квадратного кореня ...) [4].

1.3.1 Повний перебір

Найочевидніший метод вирішення задачі комівояжера - метод повного перебору або Brut-Force. Він полягає в перевірці всіх можливих варіантів шляхів, алгоритм можна записати так:

А) визначити кількість можливих гамільтонових контурів;

Б) підрахувати вагу кожного гамільтонова контуру, склавши всі ваги його ребер;

В) обрати гамільтоновий контур, вага якого мінімальни, який і буде оптимальним.

Метод повного перебору має свої переваги - він гарантує знаходження рішення задачі, при цьому він простий у виконанні. У той же час, алгоритм

вважається неефективним при роботі з великим об'ємом даних, тому що для знаходження найкоротшого шляху вимагає знайти ваги (n- 1)! гамільтонових контурів [2].

1.3.2 Метод гілок і меж

Метод гілок і меж використовують для знаходження оптимального рішення задач комбінаторної оптимізації. Він полягає в розбитті множини на підзадачі і виключення свідомо неоптимальні рішень.

Метод працює на дереві рішень та визначає принципи роботи конкретних алгоритмів пошуку розв’язків, іншими словами, є мета-алгоритмом. Для різних задач комбінаторної оптимізації створюють спеціалізовані алгоритми гілок та меж [5].

Результатом роботи алгоритму є знаходження максимуму функції на допустимій множині. При чому множина може бути як дискретною, так і раціональною. В ході роботи алгоритму виконується дві операції: розбиття вихідної множини на підмножини(гілки), та знаходження оцінок(меж). Існує оцінка множини згори та оцінка знизу. Оцінка згори — точка що гарантовано не менша за максимум на заданій підмножині. Оцінка знизу — точка що гарантовано не більша за мінімум на заданій підмножині. Множина що має найбільшу оцінку зверху зветься рекордною. На початку вся множина вважається рекордною. Рекордна множина розбивається на підмножини [5].

Результатом роботи є значення між оцінкою згори та знизу для рекордної множини. Точністю є різниця між верхньою та нижньою оцінками, тобто для дискретних множин алгоритм завершений тоді, коли ці оцінки збігаються [5].

1.3.3 Алгоритм Форда-Бельманна

Історія алгоритму зв’язана відразу з трьома незалежними математиками: Лестером Фордом, Річардом Беллманом і Едвардом Муром. Форд і Беллман опублікували алгоритм у 1956 і 1958 роках відповідно, а Мур зробив це в 1957 році. І деколи його називають алгоритмом Беллмана-Форда-Мура. Метод використовується в деяких протоколах дистанційно-векторної маршрутизації, наприклад в RIP (Routing Information Protocol- Протокол маршрутної інформації) [6].

Алгоритм Форда-Беллмана прораховує у зваженому графі найкоротші шляхи від однієї вершини до всіх інших. Він підходить для роботи з графами навіть з ребрами з від’ємною вагою. Але спектр використання алгоритму зачіпає не всі такі графи. Це пояснюється тим, що кожний наступний прохід по шляху, складеному з ребер, сума ваги яких є від’ємна, лише покращує задане значення. Нескінченна кількість покращень робить неможливим визначення одного конкретного значення, яке являлося б оптимальним. У зв’язку з цим алгоритм Форда-Беллмана не використовується з графами, у яких є від’ємні цикли, але він дозволяє визначити чи є такі[6].

1.3.4 Алгоритм Флойда-Уоршелла

Алгоритм Флойда-Уоршелла — алгоритм динамічного програмування для знаходження найкоротших шляхів між усіма вершинами зваженого орієнтованого графа. Розроблений в 1962 році Робертом Флойдом і Стівеном Уоршеллом [6].

Динамічне програмування – це альтернативне вирішення задач методом «в лоб», тобто brute forc'ом або жадібним алгоритмом. Використовується там, де оптимальне вирішення підзадачі меншого розміру може бути використано для вирішення основної задачі. В загальному виді метод виглядає так:

– розбір задачі на під задачі меншого розміру;

– знаходження оптимального вирішення під задач рекурсивно;

– використання результатів під задач для вирішення основної задачі.

Для знаходження найкоротших шляхів між всіма вершинами графа використовується не перебір всіх можливостей, що приведе до використання часу та затрат більшого об’єму пам’яті, а динамічне програмування, тобто під задачі, які будуть потрібні для вирішення головної задачі, прораховуються завчасно і потім використовуються [6].

1.3.5 Алгоритм Дейкстри

Задача про найкоротший шлях полягає у знаходженні найкоротшої відстані від заданої початкової вершини до заданої кінцевої вершини. Формулювання задач про знаходження відстаней так:

– для заданої початкової вершини *a* знайти найкоротші шляхи від *a* до всіх інших вершин;

– знайти найкоротші шляхи між усіма парами вершин [6].

Майже всі способи розв’язання задачі про найкоротший шлях від заданої сартової вершини до заданої кінцевої вершини, дають змогу знайти й найкоротші шляхи від стартової вершини до всіх інших вершин графа. Це означає, що з їх допомогою можна розв’язати задачу 1 із невеликими додатковими обчислювальними витратами. З іншого боку, задачу 2 можна розв’язати або *n* разів застосувавши алгоритм задачі 1 із різними початковими вершинами, або один раз застосувавши спеціальний алгоритм [6].

Алгоритм Дейкстри – найбільш ефективний з існуючих алгоритм на графах винайдений вченим Е. Дейкстром у 1959 році. Цей алгоритм шукає найкоротшу відстань від однієї вершини графу до всіх інших. Алгоритм працює лише для графів у яких ребра не мають від’ємної ваги. Алгоритм широко застосовується в програмуванні і технологіях [6].

1.3.6 Алгоритм A*

Алгоритм пошуку A* використовуютть для пошуку оптимального шляху між двома вершинами графу з додатніми вагами ребер. Описаний 1968 р. Пітером Хартом, Нільсом Нільсоном та Бертрамом Рафаелєм [8].

Даний алгоритм належить до евристичних алгоритмів пошуку, оскільки порядок обходу вершин графа при його реалізації передбачає першочерговість аналізу тих вершин, які ймовірно формують найкоротший шлях.

Пошук таких вершин здійснюється шляхом співставляння кожній, суміжній з поточною, вершині $v \in V$значення функції

$$f(v) = g(v) + h(v),$$

де g(v) - функція, значення якої дорівнює довжині найкоротшого шляху від початкової вершини до вершиниv; h(v) - евристична функція, яка оцінює вартість шляху від вершиниv до кінцевої вершини.

Вершини з найменшим значенням f(v)обираються в першу чергу.

Прикладом евристики h(v)може бути відстань від v до кінцевої вершини по прямій лінії, оскільки це фізично найменша можлива відстань між двома точками.

Алгоритм ділить вершини на три класи:

− невідомі вершини: ці вершини ще не були знайдені. Ще не відомий шлях до них. На початку роботи алгоритму всі вершини, окрім початкової, належать до класу невідомих;

− відомі вершини (OpenList): вже відомий (можливо не оптимальний) шлях до цих вершин. Всі відомі вершини разом зі значеннями f зберігаються в списку. З цього списку вибираються, в першу чергу, найперспективніші вершини. Конкретна реалізація цього списку має істотний вплив на швидкодію алгоритму, і зазвичай має вигляд черги з пріоритетом (наприклад,

бінарна купа). На початку роботи алгоритму до відомих вершин належить лише початкова вершина;

– повністю досліджені вершини (ClosedList): до цих вершин вже відомий найкоротший шлях. Повністю досліджені вершини додаються до так званого замкненого списку, аби запобігти багаторазовому дослідженню вже досліджених вершин. Список повністю досліджених вершин на початку роботи алгоритму порожній.

Кожна повністю досліджена вершина має покажчик на попередні вершини. Завдяки цьому покажчикові можна пройти шляхом від цієї до стартової вершини.

Коли вершину x буде повністю досліджено, суміжні з нею вершини додаються до списку відомих вершин, а сама вершина додається в список повністю досліджених. Вказівники на попередню вершину встановлюються на x. Суміжні вершини, які вже знаходяться в списку повністю досліджених вершин, до списку відомих не додаються, а зворотні вказівники не змінюються. Суміжні вершини, які вже знаходяться в списку відомих, лише оновлюються (значення f та вказівник на попередню вершину), якщо знайдений до них шлях коротший за вже відомий [8].

Алгоритм зупиняється коли кінцева вершина потрапляє до списку повністю досліджених вершин. Знайдений шлях відтворюється за допомогою покажчиків на попередню вершину. Якщо список відомих вершин порожніє, то розв’язку задачі не існує і алгоритм зупиняє пошук.

Відтворений за зворотніми покажчиками знайдений маршрут починається з кінцевої вершини та прямує до стартової. Щоб одразу отримати шлях в правильному напрямі, зі стантової вершини до кінцевої, в умовах задачі слід поміняти місцями початок та кінець. Якщо шукати маршрут починаючи з кінцевої вершини, відтворений список буде починатися зі стартової вершини й прямуватиме до кінцевої.

1.4 Постановка задачі

Проведений аналіз існуючих підходів до побудови навігаційних маршрутів показав, що наявні способи реалізації даного функціоналу в існуючих мобільних програмних засобах передбачають при побудові маршруту можливість обрати на власний розсуд, як критерій оптимальності, лише найкоротшу відстань або найменший час. За замовчуванням система обирає найшвидший маршрут. Якщо існує два маршрути з однаковим часом проходження, система автоматично обирає найкоротший із них та пропонує альтернативний варіант.

Значним недоліком таких мобільних навігаторів є обмеженість вибору критеріїв оптимізації при побудові маршруту. Наприклад, відсутня можливість вибору маршруту з урахуванням якості дорожнього покриття, завантаженості доріг, якості їх освітленості, інформації про аварійні ситуації, ремонтні роботи, пробки, закриття дороги або зміну напрямку руху, допустимого відхилення від найменшого часу або найкоротшої дистанції тощо.

Тому актуальною є задача розробки метода побудови навігаційних маршрутів з можливістю урахування декількох критеріїв оптимальності, визначених користувачем.

Для вирішення даної задачі необхідно виконати комплекс наступних взаємопов‘язаних завдань:

– дослідити існуючі методи багатокритеріальної оптимізації;

– розробити математичне забезпечення для побудови багатокритеріальних навігаційних маршрутів;

– розробити мобільний навігатор, який має підтримувати функціонал побудови навігаційних маршрутів з урахуванням декількох критеріїв оптимальності;

– виконати експериментальне дослідження запропонованих рішень.

Висновки до розділу

Усі існуючі навігаційні системи не підтримують функціонал побудови маршрутів з урахуванням декількох критеріїв оптимальності. Можлива лише побудова автоматично прокладеного маршруту від поточного місця розташування до заданої точки, де критерієм оптимальності є найкоротша відстань або найменший час. У більшості навігаційних додатків початкова точка для побудови маршруту встановлюється автоматично.

Огляд алгоритмів пошуку оптимального маршруту, які використовуються у мобільних застосуваннях-навігаторах, показав, що вони підходять лише для пошуку найкоротшого шляху між двома точками, але без урахування додаткових критеріїв. Найбільш прийнятним із них з точки зору ефективності при побудові навігаційних маршрутів є алгоритм A*, оскільки він завдяки евристичній складовій скеровує напрям пошуку та скорочує його тривалість. Однак він потребує удосконалення з урахуванням необхідності реалізації багатокритеріальної оптимізації.

2 МАТЕМАТИЧНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОБУДОВИ НАВІГАЦІЙНИХ МАРШРУТІВ

Традиційним підходом до побудови навігаційних маршрутів є моделювання цифрової географічної карти (мережі доріг) у вигляді зваженого графа, вершини якого представляють собою місця можливих змін напрямку доріг (дорожні розв'язки, перехрестя, повороти дороги тощо), а ребра – самі дороги, і вирішення задачі пошуку найкоротшого шляху між заданими вершинами даної мережі.

Орієнтовані ребра у такому графі відповідають вулицям з одностороннім рухом, неорієнтовані - вулицям з двостороннім рухом. Ваги ребер можуть визначати відстані між окремими вершинами, час проїзду по ділянці дороги тощо. Відповідно, основними критеріями пошуку у такій мережевій моделі є, як правило, найкоротший шлях або найшвидший маршрут.

Необхідність урахування декількох критеріїв оптимальності одночасно зводить дану задачу до задачі багатокритеріальної оптимізації [9].

2.1 Багатокритеріальна оптимізація

Багатокритерійна оптімізація - це процес одночасної оптимізації двох або більше конфліктуючих цільових функцій в заданій області визначення [9].

Задача багатокритеріальної оптимізації - це математична модель прийняття оптимального рішення за декількома критеріями. Ці критерії можуть відображати оцінки різних якостей об'єкта або процесу, з приводу яких приймається рішення [9].

Завдання багатокритеріальної оптимізації формулюється в такий спосіб:

$$\min_{\overline{x}}\{f_1(\overline{x}), f_2(\overline{x}), \dots, f_k(\overline{x})\}, \qquad (1)$$

$$\overline{x} \in S$$

де $f_1(\overline{x}), f_2(\overline{x}), \dots, f_k(\overline{x})$ - скалярні функції векторного аргументу $\overline{x} = (x_1, x_2 \dots, x_n)^T$, кожна з яких є математичним виразом одного критерію оптимальності, *S* – множина допустимих рішень.

Вирішення даної задачі полягає в пошуку вектора цільових змінних $\overline{x}$, що задовольняє накладеним обмеженням, і оптимізує векторну функцію, елементи якої відповідають цільовим функціям $f_1(\overline{x})$, $f_2(\overline{x}), \dots, f_k(\overline{x})$.

Так як в моделі (1) використовується векторна цільова функція, таку задачу ще називають задачею векторної оптимізації [9]. Основною особливістю даної задачі, як часткового випадку багатокритеріальної оптимізації, є наявність невизначеності, яка полягає у тому, що невідомо, яким критеріям треба віддати перевагу і в якій мірі [9].

Деякий розв'язок $\overline{x}$* задачі (1) назива- ють ефективним рішенням даної задачі, якщо таке рішення не можна поліпшити по якомусь із критеріїв, не погіршивши при цьому значення інших критеріїв [10]. Множину ефективних рішень називають множиною Парето. Оптимальне рішення слід шукати лише серед елементів множини Парето [9]. Це дозволяє звузити клас можливих претендентів на остаточне рішення і виключити з розгляду завідомо неконкурентоспроможні варіанти [9].

Ухвалення рішення - це вибір альтернативи, яка одночасно задовольняє і нечітким цілям, і нечітким обмеженням. У цьому сенсі, цілі і обмеження є симетричними щодо рішення, що стирає відмінності між ними і дозволяє представити рішення як злиття нечітких цілей і обмежень [11].

Принцип Парето. Він широко використовується при ранжируванні варіантів рішень, об'єктів і т.п. Стан А (безліч параметрів) називається Парето-оптимальним, якщо не існує іншого стану В (безлічі інших параметрів) домінуючого стан А щодо цільової функції. Стан А домінує стан В, якщо хоча б по одному параметру А краще В, а по решті не гірше.

Стосовно до задачі переговорів цей принцип стверджує що, якщо для ситуації В існує така ситуація А, що виграш кожного з учасників переговорів при реалізації ситуації А чи не менше, ніж при реалізації ситуації В і, принаймні, один парламентер отримає виграш строго більший, то вони вважатимуть за краще ситуація А ситуації В [12].

Тому термін «оптимізувати» в задачі багатокритеріальної оптимізації означає: знайти таке рішення, при якому значення цільових функцій $f_1(\overline{x}),\ f_2(\overline{x}), \dots,\ f_k(\overline{x})$ були б прийнятними для постановника задачі.

Існуючі підходи до вирішення задачі багатокритеріальної оптимізації передбачають використання:

– методів, заснованих на згортанні критеріїв;

– методів, які використовують обмеження на критерії;

– методів цільового програмування;

– методів, заснованих на знаходженні компромісного рішення [9].

Замість вихідної багатокритеріальної задачі відповідно до обраного методу, формується задача, що її заміщає. До складу такої задачі входить один критерій, а до вихідної системи обмежень додається одне або декілька додаткових обмежень. Розв'язок такої задачі називається субоптимальним [9].

2.1.1 Метод ідеальної точки

Метод використовує множину Парето, яка у даному випадку складається з допустимих точок завдання, які не можуть бути «зрушені» в межах допустимої множини з поліпшенням відразу за обома критеріями.

Метод ідеальної точки полягає у знаходженні на границі Парето точки, найближчої до точки утопії, що задається особою, що приймає рішення (ОПР). Зазвичай ОПР формулює мету у вигляді бажаних значення показників, і часто як координати цільової точки вибирається поєднання найкращих

значень всіх критеріїв (зазвичай ця точка не реалізується при заданих обмеженнях, тому її і називають точкою утопії).

Ідеальна точка визначається як вектор $y^I = (y_1^I, \dots, y_p^I)$, кожна з координат якого має оптимальне значення відповідної складової цільової функції:

$$y_k^I = \min_{x \in X} f_k(x) = \min_{y \in Y} y_k.$$

Утопічну точку y^U обчислюють на основі ідеальної:

$$y^U = y^i - \epsilon U$$

де $\epsilon > 0, U$ — одиничний вектор [13].

2.1.2 Метод послідовних поступок

Процедура розв'язування багатокритеріальної задачі методом послідовних поступок полягає у тому, що часткові критерії нумерують у порядку їхньої відносної важливості; максимізують перший, найважливіший критерій; потім призначають величину припустимого зниження значення цього критерію і максимізують другий за важливістю частковий критерій за умови, що значення першого критерію не повинно відрізнятиметься від максимального більш ніж величину встановленого зниження (поступки); знову призначають величину поступки, але вже другому критерію і знаходять максимум третього за важливістю критерію за умови, щоб значення у перших двох критеріїв не відрізнялися від раніше знайдених максимальних значень більш як на величини відповідних поступок; далі аналогічним чином по черзі використовуються й інші часткові критерії; оптимальною зазвичай вважають будь-яку стратегію, яка отримана під час розв'язування задачі пошуку умовного максимуму останнього за важливістю критерію. Отже, під час використання методу послідовних поступок багатокритеріальну задачу зводять до почергової максимізації часткових критеріїв і вибору величин поступок. Величини поступок характеризують відхилення пріоритету перших

приватних критеріїв над іншими від лексикографічного: чим менші поступки, тим жорсткіший пріоритет [13].

2.1.3 Метод згортання критеріїв

Метод згортання критеріїв – приведення множини критеріїв до одного глобального за допомогою вагових коефіцієнтів, які відображають важливість критеріїв, та розв'язування класичної однокритеріальної задачі [14].

Основним недоліком даного методу є проблема виявлення точних значень вагових коефіцієнтів, процедура визначення яких в більшості випадків не є об'єктивною. Також коефіцієнти в методі згортання повинні бути розмірними, тому що критерії в більшості випадків мають різну розмірність. З метою позбавлення цієї проблеми окремі критерії нормуються (нормовані критерії є безромірними та змінюються в інтервалі від 0 до 1). Але нормовані критерії не мають змістовного навантаження, тому об'єктивний підрахунок вагових коефіцієнтів ускладнюється ще більше[14].

2.2 Математична постановка задачі

Необхідно визначити маршрут мінімальної довжини між двома вершинами і всі близькі до нього в заданому діапазоні відхилення довжини від мінімальної.

Математично постановка даної задачі може бути сформульована наступним чином:

Нехай заданий зв'язний зважений граф $G = (V, R)$, вершини V якого відповідають вузлам цифрової географічної карти (місцям можливих змін напрямку доріг - перехрестям, поворотам дороги тощо), ребра R – шляхам, що з'єднують ці вузли, а ваги ребер w – протяжностям цих шляхів.

Також кожне ребро $r = (u, v) \in R,\ u \in V, v \in V$ має свій коефіцієнт якості $k(u, v) \in K,\ k = (0, 1]$, де K – це множина коефіцієнтів якості ребер графа *G*.

Маршрутом у графі є послідовність ребер $r \in R$, які утворюють неперервну послідовність вузлів:

$$\forall r_i \in R: end(r_{i-1}) = start(r_i),$$
$$end(r_i) = start(r_{i+1}),$$

де $start(r) \in U$ - це номер вузла, з якого виходить ребро r, $end(r) \in U$ - це номер вузла, в яке виходить ребро r.

Необхідно визначити маршрут мінімальної довжини між двома заданими вершинами, а також усі альтернативні маршрути між цими ж вершинами, відхилення довжин яких від мінімальної знаходиться в заданому діапазоні.

2.2.1 Математичний метод знаходження оптимального маршруту

Задача побудови навігаційних маршрутів з можливістю урахування декількох критеріїв оптимальності, як задача мережевого аналізу, передбачає моделювання системи (у нашому випадку - цифрової географічної карти) у вигляді графа та подальший аналіз даного графа з метою пошуку найкоротшого (найоптимальнішого за вказаними критеріями) маршруту між двома його вершинами.

Оскільки дана задача передбачає можливість урахування декількох критеріїв оптимальності, то, щоб враховувати цю інформацію при аналізі мережі, треба розглядати зважені графи.

Зваженим є граф, кожному ребру якого ставиться у відповідність число, яке називається вагою цього ребра.

Для того, щоб побудувати навігаційний маршрут з урахуванням декількох критеріїв оптимальності, пропонується за допомогою методів

багатокритеріальної оптимізації перезважити ребра графу з урахуванням вказаних критеріїв.

Для вирішення даної задачі можна використати метод згортання критеріїв.

Відповідно до даного методу, багатокритеріальна задача зводиться до задачі з одним зваженим критерієм (2):

$$\min_{\overline{x \in X}} \sum_{k=1}^{p} \lambda_k f_k(x), \qquad (2),$$
$$\sum_{k=1}^{p} \lambda_k = 1,$$
$$\lambda_k \geq 0,$$

де p – кількість критеріїв, f – функція відповідного критерію, λ - ваговий коефіцієнт відповідного критерію, X – множина допустимих рішень при різних значеннях λ_k.

Результуючий ваговий коефіцієнт якості кожного ребра графа при цьому підраховується за формулою (3):

$$k(r) = \sum_{k=1}^{p} \lambda_k f_k(r). \qquad (3)$$

З урахуванням вищезазначеного, результуючою вагою ребра *Wi* між двома вершинами графа є добуток отриманого вагового коефіцієнту якості ребра$r_i \in R$, що визначаються за заданими критеріями, на довжину даного ребра *w*(4):

$$Wr_i = w(r_i)k(r_i), r_i \in R, \qquad (4)$$

де *i* – індекс ребра.

Результуючою вагою маршруту W_j між двома довільними вершинами графа є сума ваг кожного ребра Wr_i даного шляху:

$$W(R_j) = \sum_i W(r_i), \qquad (5)$$

де *j*– індекс маршруту, R_j – це множина ребер r_i, що утворюють маршрутW_j.

Згідно метода згортання критеріїв, оптимальним маршрутом до будь-якої вершини графу є шлях з мінімальною вагою (6):

$$W(R_o) = \sum_i w(r_i)k(r_i) \rightarrow min,$$

$$\forall R_j \in T,\ W(R_o) = \underset{R_j \in T}{\min} W(R_j) = \min_{\overline{R_j \in T}} \sum_i w(r_i) \sum_{k=1}^{p} \lambda_{ik} f_{ik}(r_i),$$

де R_o – це множина ребер r_i, що утворюють маршрут мінімальної довжини, T – множина допустимих маршрутів.

2.2.2 Алгоритм знаходження найкоротшого шляху.

Для визначення маршруту мінімальної довжини пропонується застосовувати удосконалений варіант відомого алгоритму A*[15], що належить до евристичних алгоритмів пошуку.

Алгоритм A* заснований на розстановці позначок на вершинах графа, які представляють собою скалярні величини, що дорівнюють сумі вагових коефіцієнтів ребер графа на шляху від початкової точки до поточної.

Модифікація даного алгоритму полягає у тому, що на кожній ітерації процесу пошуку необхідно не намагатися вибрати значення позначки з меншим значенням, що відповідає поточній вершині графа, а створювати нову позначку, доповнюючи нею множину позначок поточної вершини. Вихідними даними модифікованого алгоритму буде множина маршрутів, яку можна назвати множиною субоптимальних маршрутів.

Так як алгорим працює з «переваженим графом» в залежності від критеріїв оптимальності, необхідно перерарухавати значення оригінальної довжини кожного маршруту та порівняти їх з найкоротшим. Оригінальна довжина субоптимальних маршрутів не повинна перевищувати максимально допустиму похибку - ε, що є вхідним параметром алгоритму.

Алгоритм пошуку субоптимальних маршрутів знаходить альтернативні шляхи, довжини яких перевищують довжину найкоротшого маршруту в межах допустимого відхилення ε:

$$\forall R_S \in T, \frac{W(R_o)}{W(R_j)} < \varepsilon \qquad (7)$$

де $R_S -$ субоптимальний маршрут.

Представимо алгоритм пошуку субоптимальних шляхів на прикладі графу, показаного на рисунку 2.1, де початкова точка – вершина 1, кінцева точка – вершина 4, допустима похибка – 20%.

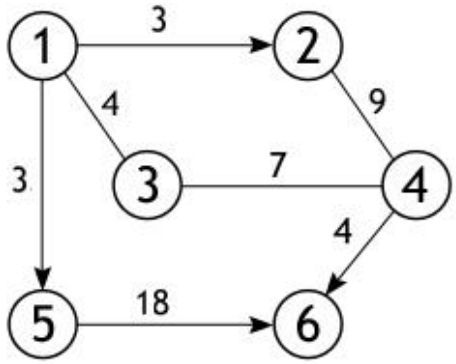

Рисунок 2.1 – Граф для представлення алгоритму

Введемо 2 критерія якості для ребер даного графа. Нехай перший критерій має пріоритет 0.7, а другий відповідно 0.3.

Задамо значення критеріїв для кожного ребра графа.

Таблиця 2.1 – Критерій 1

1-2	2-4	1-3	3-4	1-5	5-6	4-6
0.25	1	1	1	0.7	0.23	0.32

Таблиця 2.2 – Критерій 2

1-2	2-4	1-3	3-4	1-5	5-6	4-6
0.53	1	1	1	0.6	0.94	0.92

Крок 1. Підрахуємо вагові коефіцієнти для кожного ребра за формулою (3) та ваги ребер з урахуванням цих критеріїв за формулою (4) (рисунок 2.2).

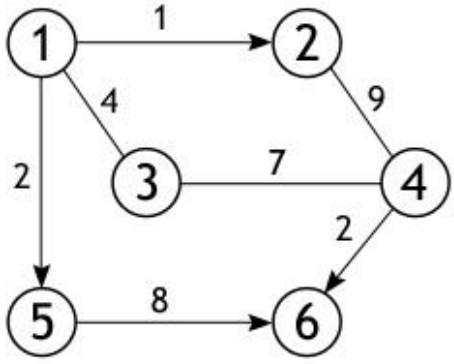

Рисунок 2.2 – Стан графа на 1 кроці алгоритму

Крок 2. Початковій вершині 1 встановимо позначку із значенням 0.

Крок 3. У вершини 1 рівно три сусіда: вершини 2, 3, 5. Щоб обчислити довжину шляху до кожного з них потрібно скласти ваги ребер, що лежать між вершинами 1 і 2, 1 і 3, 1 і 5 зі значенням позначки першої вершини (тобто, з 0) (рисунок 2.3):

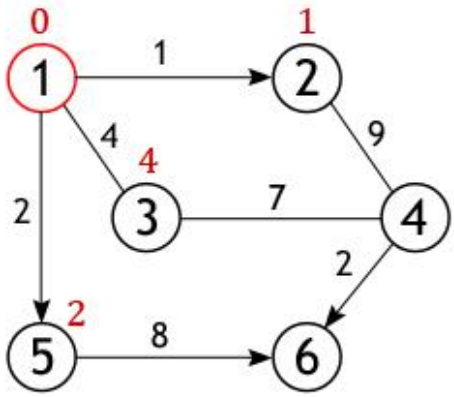

Рисунок 2.3 – Стан графа на 3 кроці алгоритму

Крок 4. Поточна (активна) вершина позначається як відвідана, статус «активної» переходить до однієї із її сусідок, а саме до вершини 2, оскільки вона найближча по довжині ребра до раніше активної вершини (рисунок 2.4).

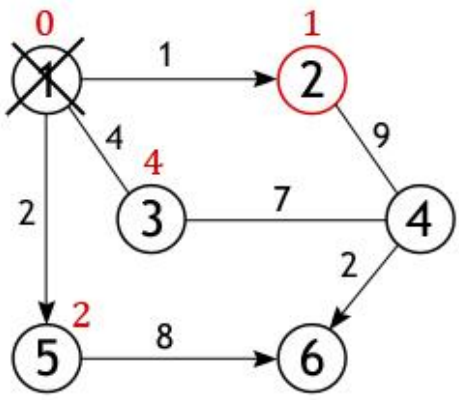

Рисунок 2.4 – Стан графа на 4 кроці алгоритму

Крок 5. У вершини 2 всього один не розглянутий сусід, відстань до якого з неї дорівнює 9. Необхідно обчислити довжину шляху з початкової вершини 1, для чого потрібно скласти величину позначки вершини 2 з вагою ребра до вершини 4. До позначок вершини 4 додається позначка зі значенням 10 (рисунок 2.5).

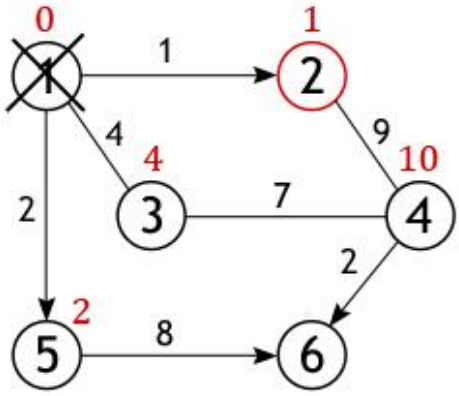

Рисунок 2.5 – Стан графа на 5 кроці алгоритму

Крок 6. Вершина 2 перестає бути активною, і видаляється зі списку не відвіданих. Тепер тим же способом досліджуються сусіди вершини 5, і обчислюється відстань до них (рисунок 2.6). У вершини 6 немає не розглянутих сусідів.

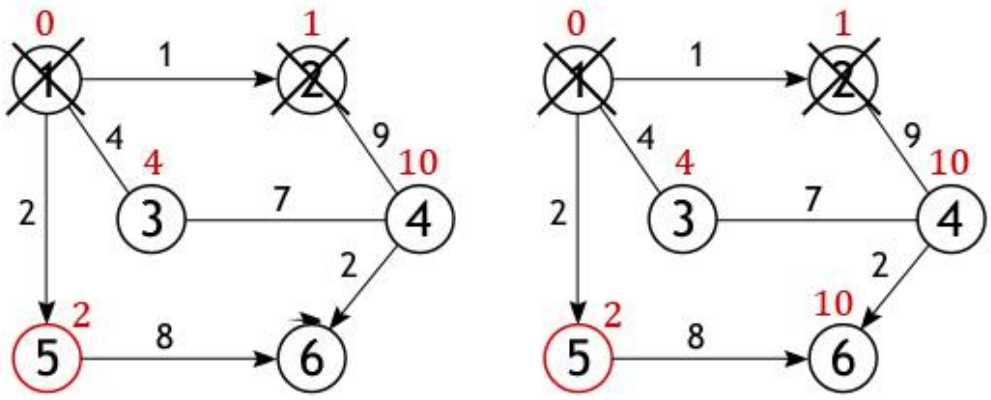

Рисунок 2.6 – Стан графа на 6 кроці алгоритму

Крок 7. Аналогічно розглядаємо вершину 3 та її сусідів. При проході через вершину 3 до кінцевої точки, вершина 4 отримує додаткову позначку – 11 (рисунок 2.7).

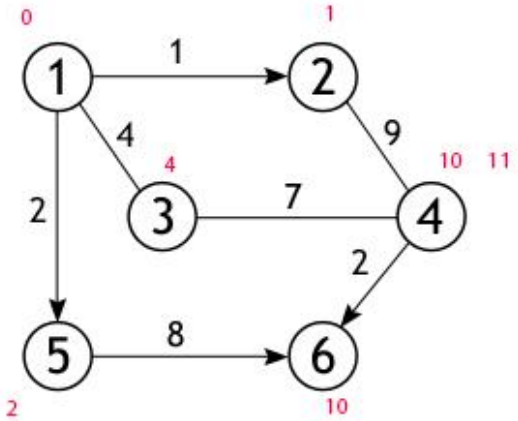

Рисунок 2.7 – Стан графа на 7 кроці алгоритму

Крок 8. З урахуванням вказаних критеріїв маємо субоптимальні маршрути: 1-2-4 (1 маршрут) з довжиною 10 та 1-3-4 (2 маршрут) з довжиною 11.

Крок 9. Підрахуємо оригінальні довжини даних шляхів:

1-2-4 = 12

1-3-4 = 11

Крок 10. Так як оригінальна довжина субоптимального маршруту 1 більша за довжину маршруту 2, але в межах допустимої похибки (20%), то вважаємо, що маршрут 1 є оптимальним по заданим критеріям.

Висновки до розділу

Запропоновано метод побудови навігаційних маршрутів з урахуванням декількох критеріїв оптимальності, що базується на використанні удосконаленого алгоритму A*.

Використання такого алгоритму дозволяє визначити декілька субоптимальних рішень, для яких необхідно підрахувати оригінальні довжини маршруту та відкинуті ті, що перевищують максимально допустиме відхилення.

3 ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ ПОБУДОВИ НАВІГАЦІЙНИХ МАРШРУТІВ

3.1 Вимоги до програмного забезпечення

У розділі 1 були розглянуті застосунки, що досить якісно вирішують задачу навігації та побудови маршруту на місцевості. За результатами аналізу виявлені недоліки, а саме відсутність можливості побудови навігаційного маршруту з урахуванням декількох критеріїв.

Для відображення роботи навігаційного додатку було обрано критерій – якість дорожнього покриття, данні про дороги можна отримати з відкритих джерел, а саме з API сайту сервісу УкрЯма.

API - набір визначень взаємодії різнотипного програмного забезпечення. API - це зазвичай (але не обов'язково) метод абстракції між низькорівневим та високорівневим програмним забезпеченням.

Отримані дані з сервісу УкрЯма необхідно відфільтрувати та кожному дефекту на дорози встановити коефіцієнт.

Є декілька типів дефектів на дорозі (рисунок 3.1):

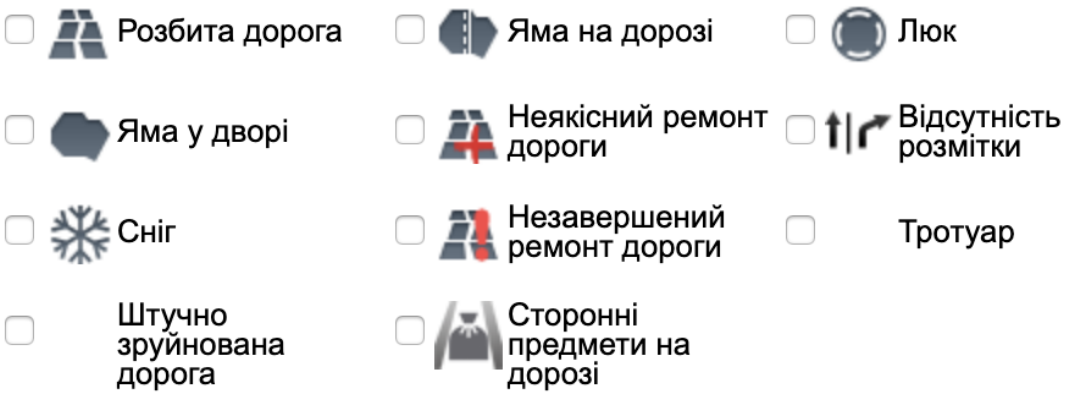

Рисунок 3.1 - Можливі дефекти на дорозі

Дефекту Розбита дорога та Штучно зруйнована дорога та Незавершений ремонт дороги встановлюється коефіцієнт 0.9. Відсутності

розмітки надано коефіцієнт 0.6. Неякісний ремонт – 0.75. Дефект яма на дорозі встановлює коефіфієнт 0.9, але для одного кілометру дороги.

Враховуючи запропоновані у розділі 1 підходи до вирішення задачі побудови багатокритеріальних навігаційних маршрутів, були сформовані функціональні та нефункціональні вимоги до реалізації програмного забезпечення. До нефункціональних вимог входять:

– розроблене програмне забезпечення представляє собою мобільний застосунок;

– розроблене програмне забезпечення працює на базі удосконаленого A* з використанням методів багатокритеріальної оптимізації.

До функціональних вимог входять:

– можливість вибору кінцевої точки маршруту;

– можливість вибору початкової точки маршруту;

– можливість вибору допустимого відхилення від мінімальної довжини маршруту;

– можливість вибору критеріїв для побудови навігаційного маршруту.

На рисунку 3.2 зображена діаграма використання, що відповідає функціональним вимогам:

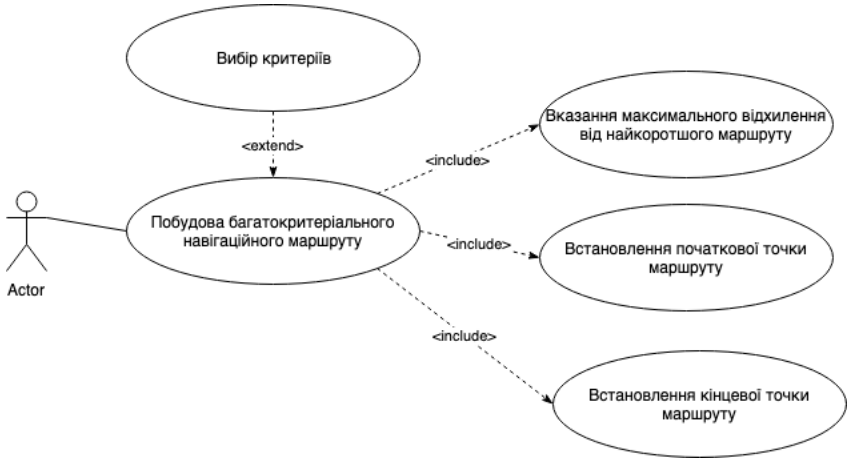

Рисунок 3.2 – Діаграма використання

Для відображення карти було обрано картографічний сервіс MapBox. Для отримання картографічних даних використовується картографічний сервіс Open Street Maps (OSM). Дані карти зберігаються у форматі XML, де теги **node** містять інформацію про об’єкти на карті та перехрестя, а теги **way** описують існуючі дороги. Структура XML файлу показана на рисунку 3.3.

OSM xml.xml | Main Form.aspx.cs | Main Form.aspx

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="OpenstreetServer">
  <node lat="30.866056900000004" lon="75.8824308">
    <tag k="amenity" v="Atm" />
    <tag k="Name" v="HDFC ATM" />
    <tag k="addr:StreetNo." v="2" />
    <tag k="addr:HouseNo." v="32" />
    <tag k="addr:City" v="jagraon" />
    <tag k="addr:state" v="Punjab" />
    <tag k="addr:country" v="Indian" />
    <tag k="PhoneNo." v="326" />
    <tag k="Website" v="www.example.com" />
    <tag k="comment" v="Adding POI" />
    <tag k="created_by" v="Er. Pawan verma" />
  </node>
</osm>
```

Рисунок 3.3 – Структура OSM

Щоб отримати файл-базу даних необхідно відправити http запит типу GET за адресою https://api.openstreetmap.org/api/0.6/map, та в параметри запиту передати географічні координати обмеженого прямокутника у форматі:

bbox=minLon,minLat,maxLon,maxLat

OpenStreetMap – міжнародний проект метою якого є створення вільної, відкритої мапи світу. Велика кількість учасників по всьому світу в режимі реального часу додаютьт дані про місцевість в якій вони живуть, допомагають наповнити мапу даними про об’єкти інфраструктури поруч з ними, приймають участь у створенні мапи для допомоги постраждалим у зонах стихійних лих, катастроф й таке інше. Дані, зібрані учасниками

проекту, є вільними, відкритими та безкоштовними. Вони отримані з відкритих джерел, передані урядовими та комерційними організаціями, а також зібрані в ході досліджень місцевості рядовими учасниками проекту. OpenStreetMap – це відкриті дані, їх можно вільно використовувати для будь-яких цілей. [16]

3.2 Опис архітектури програмного забезпечення

Мобільний додаток було розроблено за клієнт-серверною архітектурою.

Клієнт-серверна архітектура – це розподілена структура програми, яка розділяє завдання або навантаження між постачальниками ресурсу або сервісу, що називаються серверами, та запитувачами сервісів, які називаються клієнтами. Часто клієнти та сервери спілкуються через комп'ютерну мережу на окремому апаратному забезпеченні, але як клієнт, так і сервер можуть перебувати в тій самій системі. Хост сервера запускає одну або декілька серверних програм, які надають ресурсами клієнтам. Клієнт не надає ніякого ресурсу, але користується вмістом сервера або його службовими функціями. Тому саме клієнти починають комунікацію з серверами, які чекають на вхідні запити [17].

У даному випадку розроблене програмне забезпечення складається с двох частин:

- клієнтський додаток;
- бібліотека для побудови навігаційного маршруту.

Дані про якість дорожнього покриття отримуємо з сервісу УкрЯма.

На рисунку 3.4 зображений зв’язок вищеописаних компонентів розробленого програмного забезпечення.

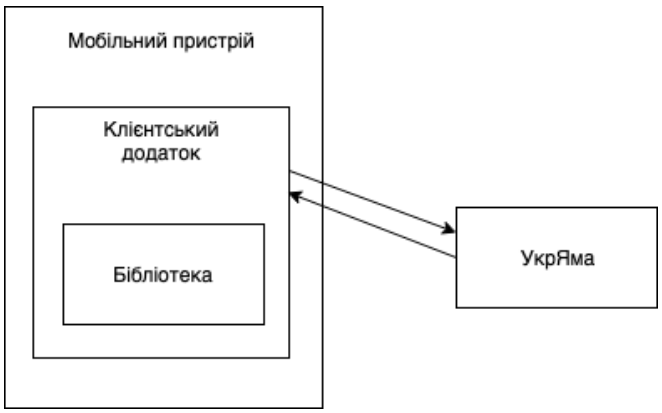

Рисунок 3.4 – Архітектура

Клієнтська частина написана на мові Swift, розроблена за архітектурою MVC.

Swift - це об'єктно-орієнтована мова програмування, розроблена компанією Apple для того, щоб співіснувати з Objective C і бути стійкішою до помилкового коду. Swift була представлена на конференції розробників WWDC 2014. Мова побудована з LLVM компілятором, включеного у Xcode 6 beta [18].

Компілятор Swift побудований з використанням технологій вільного проекту LLVM. Swift успадковує кращі елементи мов C і Objective-C, тому синтаксис буде звичний для знайомих з ними розробників, але водночас відрізняється використанням засобів автоматичного розподілу пам'яті і них і масивів, що значно збільшує надійність і безпеку коду [18].

При цьому Swift-програми компілюються у машинний код, що дозволяє забезпечити високу продуктивність. За заявою Apple, код Swift виконується в 1.3 рази швидше коду на Objective-C. Замість збирача сміття Objective-C в Swift використовуються засоби підрахунку посилань на об'єкти, а також надані у LLVM оптимізації, такі як автовекторизація [18].

MVC - архітектурний шаблон, який використовується під час проектування та розробки програмного забезпечення [19].

На Рис. 3.5 зображена архітектура MVC:

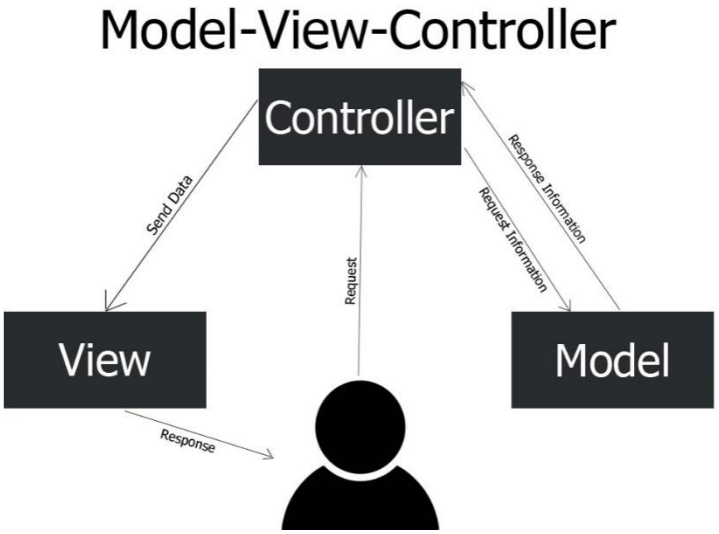

Рисунок 3.5 – Архітектура MVC

У даному випадку клієнтський додаток складається з складається з таких модулів, як:

– карта;

– вибір критеріїв;

– бібліотека для побудови багатокритеріальних навігаційних маршрутів;

– API-інтерфейс для роботи з сервісом УкрЯма.

Додаток був розроблений для мобільної платформи iOS версії 9.0 та вище.

iOS (відома як iPhone OS до червня 2010 року) – це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, вона стала операційною системою також для iPod Touch, iPad і Apple TV. Apple не дозволяє роботу ОС на мобільних телефонах інших фірм [20].

iOS є похідною від OS X, отже, є за своєю природою Unix-подібною операційною системою [20].

Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу

управління складаються з повзунків, перемикачів і кнопок. Він призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертання пристрою у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами [20].

Мобільний додаток був розроблений за допомогою об'єктно-орієнтовної мови програмування Swift. Детальний опис методів класів наведено у таблиці 3.1.

Таблиця 3.1 – Опис класів

Клас	**Опис**
AppDelegate	Стандартний клас, що містить методи для оповіщення про зміну стану додатку
Vertex	Клас, за допомогою якого описується граф та характеризуються вершини
Astar	Клас, відповідає за побудову субоптимальних маршрутів
MapDataLoader	Клас, що завантажує дані про карту
MapDataParcer	Клас, що конвертує дані карти в граф
UkrYamaLoader	Клас, що завантажує дані про дефекти на дорозі
UkrYamaParcer	Клас, що парсить дані про дороги та підраховує коефіцієнти оптимальності
LatLon	Модель географічних координат
Utils	Клас з утилітами
RequestHelper	Клас, що керує менеджерами для виконня http запитів

Продовження таблиці 3.1

RequestManager	Інтерфейс, що описує функціональну складову http запитів
URLBuilder	Клас, що відповідає за побудову URL в залежності від ендпоінтів
AlamofireRequest	Менеджер http запитів, побудований на бібліотеці Alamofire
NetworkError	Універсальна модель помилки завантаження http запиту
MapViewController	Клас, що відповідає за відображення карти та маршрутів на карті
SettingsViewController	Клас, за допомогою якого можна встановити допустиму похибку та обрати критерії побудови багатокритеріального маршруту

Таблиця 3.2 − Опис методів класів

Клас	**Назва**	**Вхідні параметри**	**Опис**
AppDelegate	applicationDidFinishLaunchingWithOptions	application : UIApplication launchOptions : [UILaunchOptionsKey : Any]	Метод викликається коли додаток повністю завантажився і готовий до роботи
AppDelegate	applicationDidEnterBackground	application : UIApplication	Метод викликається коли користувач згортає додаток

Продовження таблиці 3.2

AppDelegate	applicationDidBecomeActive	application : UIApplication	Метод викликається коли користувач розгортає додаток
AppDelegate	applicationWillTerminate	application : UIApplication	Метод викликається коли користувач закриває додаток
AStar	shortestPathes	startVertex: Vertex finishVertex: Vertex delta: Double	Метод знаходить усі субоптимальні маршрути
MapDataLoader	load	startCoods: LatLon endCoods: LatLon	Метод завантажує дані про кару згідно обраного діапазону
MapDataParcer	parce	MapData	Конвертує дані про карту у граф
UkrYamaLoader	load	startCoods: LatLon endCoods: LatLon	Метод завантажує дані про дороги згідно обраного діапазону
UkrYamaParcer	parce	data: UrkYamaData graph: Vertex	Підраховує вагові коефіцієнти для ребер графа в залежності від стану доріг
Utils	widthForLabel	text : String font : UIFont	Підраховує ширину віджету з текстом

Продовження таблиці 3.2

Utils	heihtForLabel	text : String font : UIFont width : CGFloat	Підраховує висоту віджету з текстом
Utils	isIpad		Перевіряє, чи пристрій належить до сімейства iPad
Utils	getBackgroundColor		Повертає колір фону додатку
Utils	getMainColor		Повертає основний колір додатку
Utils	getStoryboard		Повертає сторіборд
Utils	getWaitViewController		Повертає вікно очікування
Utils	showAlert	title : String message : String	Показує повідомлення
Utils	removeAllUserDefaults		Видаляє усі збережені дані в додатку
RequestHelper	setupRequestManager	requestManager: RequestManager	Встановлює загальнодоступний менеджер http запитів

Продовження таблиці 3.2

AlamofireRequest	Request	requestModel: Encodable url: URL type: RequestHttpMethod headers: [String: String] completion: {SuccessModel, ErrorModel, ErrorCode}	Виконує запит до http серверу
URLBuilder	build	endpoint: Endpoint	Будує URL в залежності від ендпоінтів
URLBuilder	build	url: String	Будує URL за строковим записом

3.3 Функціональність програмного забезпечення

Розроблений мобільний застосунок представляє собою односторінковий додаток, на єдиній сторінці якого розташований весь функціонал сервісу, що робить його простим у використанні.

Функціонал мобільного додатку передбачає:

– перегляд карти місцевості;

– можливість побудувати маршрут між двома точками з урахуванням критеріїв;

– можливість вибору критеріїв для побудови маршруту;

– встановлення максимально допустимого відхилення від найкоротшого маршруту;

– відображення маршруту на карті.

На рисунках 3.6-3.11 представлені екрани додатку при використанні кожного з пунктів функціоналу відповідно:

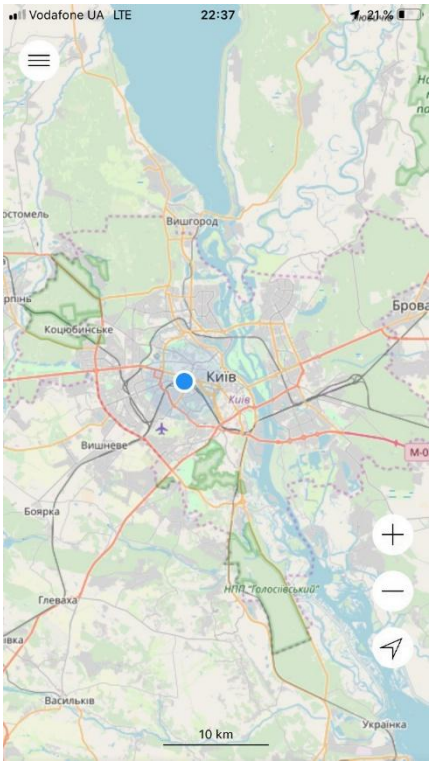

Рисунок 3.6 – Початок роботи

Рисунок 3.7 – Встановлення початкової точки маршруту

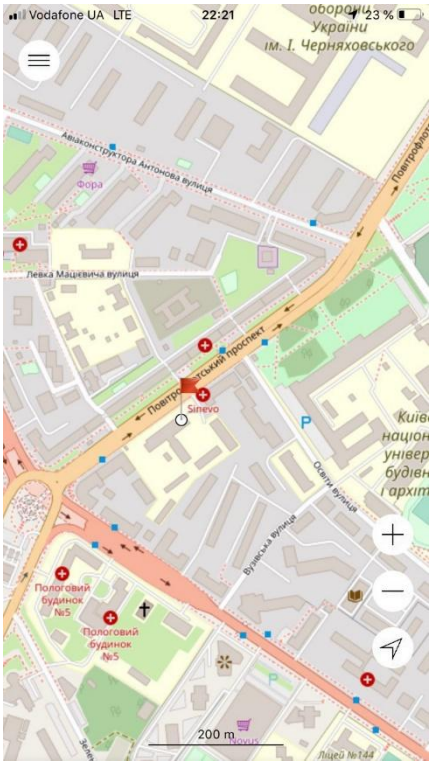

Рисунок 3.8 – Встановлення кінцевої точки маршруту

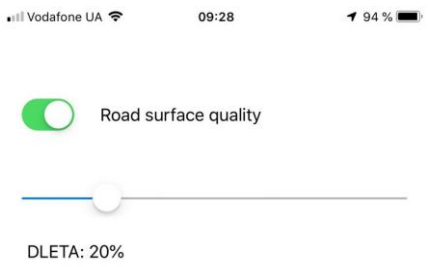

Рисунок 3.9 – Вибір критерії маршруту

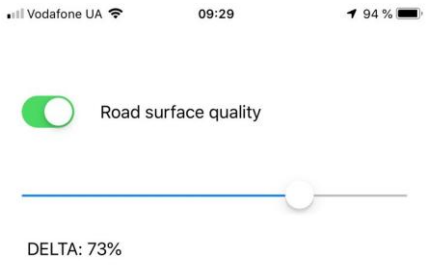

Рисунок 3.10 – Встановлення максимально допустимого відхилення від найкоротшого маршруту

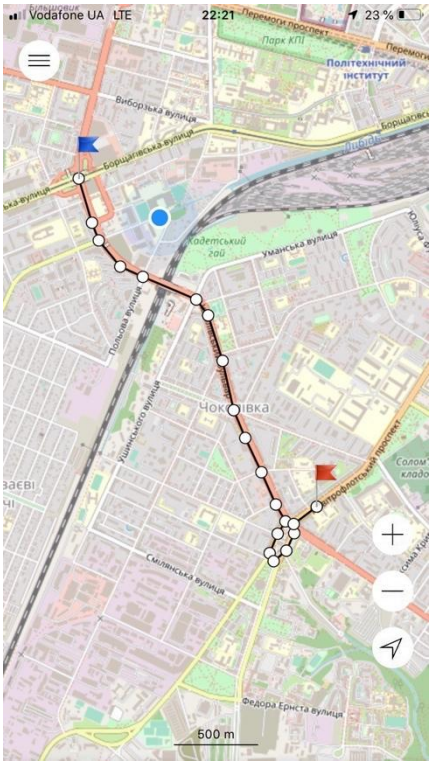

Рисунок 3.11 – Маршрут

Також алгоритмічна частина надає режим тестування. В цьому режимі додаток виводить у термінал усі можливі маршрути від початкової до кінцевої точки.

Висновки по розділу

У даному розділі представлено реалізацію розробленого програмного забезпечення, яке відповідає сформованим вимогам, а саме: є мобільним додатком, який підтримує функціонал побудови багатокритеріального навігаційного маршруту між двома точками за алгоритмом описаного у розділі 2 даної роботи, а також надає користувачу можливість задавати критерії та вказувати максимально допустиме відхилення від найкоротшого маршруту.

Розроблена бібліотека програмних засобів, яка містить набір функцій, що підтримують реалізацію, запропонованих у розділі 2, алгоритмів побудови багатокритеріальних навігаційних маршрутів.

Також мобільний додаток підтримує режим тестування, що надає можливість проаналізувати ефективність алгоритму, описаного у даній роботі.

4 РЕЗУЛЬТАТИ ЕКСПЕРИМЕНТАЛЬНИХ ДОСЛІДЖЕНЬ

4.1 Мета та порядок досліджень

Для випробовування мобільного додатку та порівняння результатів з існуючими навігаторами, було вирішено провести ряд досліджень.

Користувач має встановити початкову та кінцеву точку на карті та побудувати маршрут між ними. Після цього порівняти якість дорожнього покриття на ділянках побудованих маршрутів, опираючись на данні сервісу УкрЯма.

4.2 Дослідження побудови маршрутів

Порівняємо результати роботи існуючих навігаційних додатків з розробленим мобільним додатком, що при побудові навігаційного маршруту враховує додатковий критерій, а саме якість дорожнього покриття.

Для прикладу побудуємо маршрут від вул. Нижньоключова до станції метро Політехнічний інститут.

Результати роботи Google Maps показано на рисунку 4.1

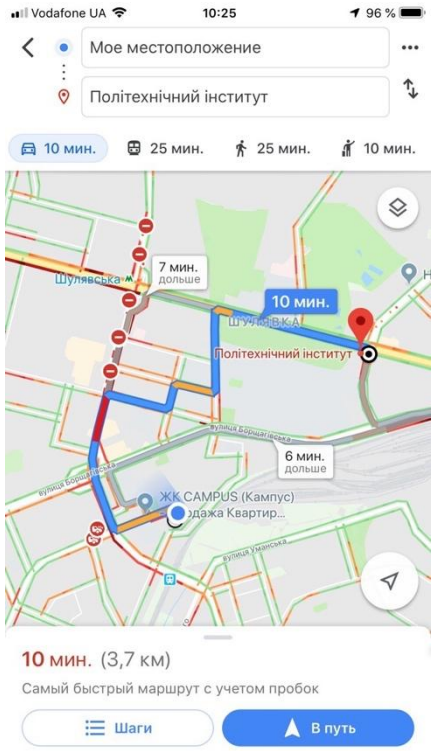

Рисунок 4.1 - Результати роботи Google Maps

Результати роботи Yadex Maps показано на рисунку 4.2.

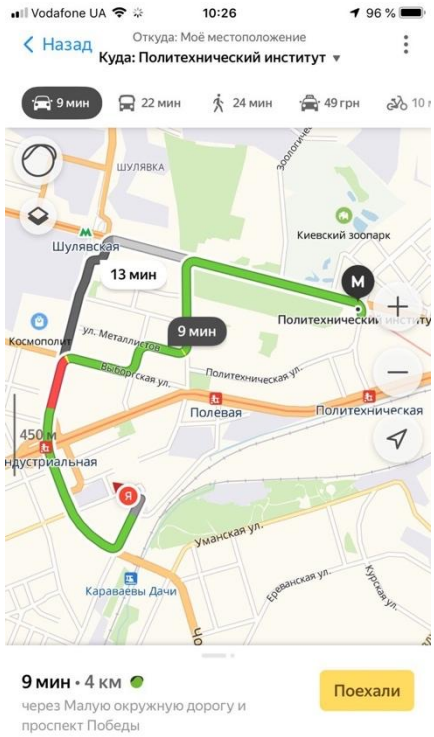

Рисунок 4.2 - Результати роботи Yandex Maps

Результати роботи розробленого навігаційного додатку показано на рисунку 4.3.

Рисунок 4.3 - Результат роботи розробленого додатку

Існуючі навігатори запропонували декілька варіантів шляху, але при побудові маршруту вони не враховували важливий критерій – якість дорожнього покриття. Аналоги запропонували проїхати по вул. Виборзькій, на якій дуже неякісне дорожнє покриття, що зазначене в сервісі УкрЯма (рисунок 4.4).

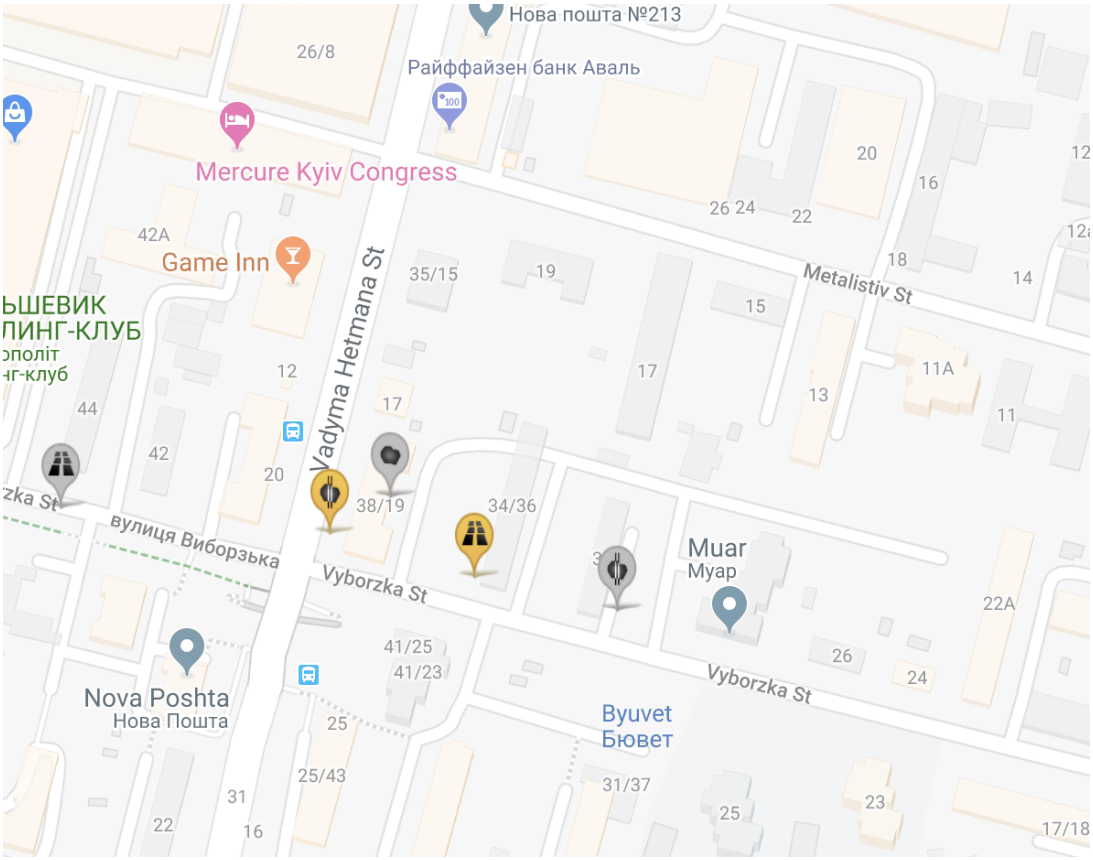

Рисунок 4.4 - Карта з дефектами (УкрЯма)

Якщо врахувати, що по усім дорогам можна рухатися з однаковою константною швидкістю, а додаткові критерії впливають на зміну швидкості в меншу сторону, то коефіцієнти якості помножені на критерії можна назвати «часом руху». Порівняння «часу» різних навігаційних додатків з додатком реалізованому у даній роботі наведено на рисунку 4.5 та у таблиці 4.1:

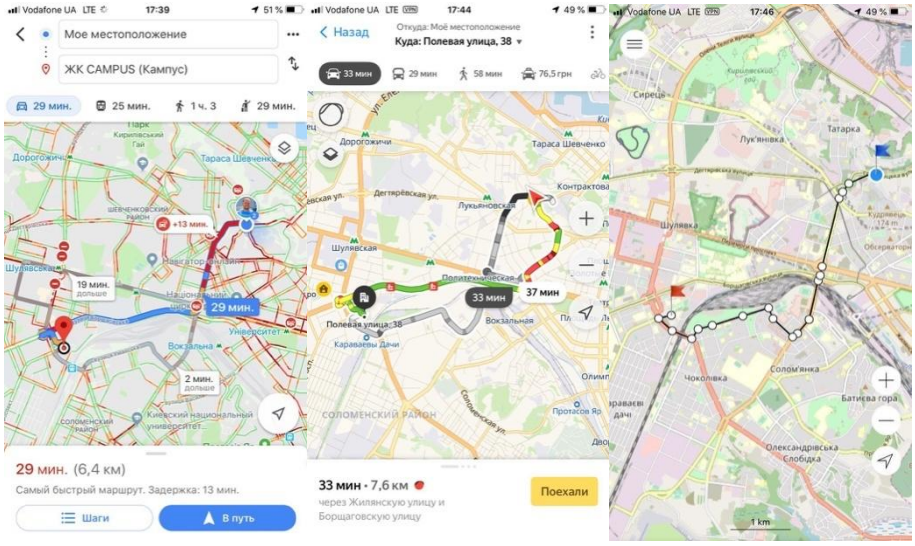

Рисунок 4.5 Порівняння результатів роботи різних навігаційних додатків

Таблиця 4.1 - Порівняння результатів роботи різних навігаційних додатків

	Відстань	«Час»
Google Maps	6.4 км	3.234
Yandex Maps	7.6 км	3.741
Навігаційний додаток розроблений в рамках цієї роботи	7.1 км	3.12

Маршрут з найкоротшою дистанцією руху побудувала Google Maps, але додаток написаний в рамках цієї роботи показав найкращий результат по «часу», причому відстань не перевищу максимально допустиму похибку – 20%.

Проведемо експеримент ще декілька разів. Результати експериментів наведено у таблицях 4.2 – 4.4.

Таблиця 4.2 – Експеримент 2

	Відстань	«Час»
Google Maps	12.2	6.1
Yandex Maps	12.2	6.1
Навігаційний додаток розроблений в рамках цієї роботи	11.4	5.985

Таблиця 4.3 – Експеримент 3

	Відстань	«Час»
Google Maps	27.5	18.654
Yandex Maps	23.6	17.341
Навігаційний додаток розроблений в рамках цієї роботи	28.3	17.211

Таблиця 4.4 – Експеримент 4

	Відстань	«Час»
Google Maps	66.9	44.153
Yandex Maps	66.9	44.153
Навігаційний додаток розроблений в рамках цієї роботи	66.9	44.153

В останньому варіанті усі навігатори побудували однакові маршрути.

4.3 Аналіз результатів досліджень

По швидкодії аналоги мобільних навігаторів виграють у розробленого програмного засобу за умови наявності гарного Інтернет з’єднання. Розроблений програмний додаток будує маршрут на пристрої користувача, Інтернет з’єднання необхідне лише для отримання динамічних даних карти та даний про стан доріг.

Після проведення ряду експериментів та внесення результатів у порівняльну таблицю можна побачити, що в деяких випадках маршрути побудовані і різних додатках співпадають.

Зустрічаються випадки, коли розроблений в рамках цієї роботи навігаційний додаток показав кращий результат навіть по довжині шляху. Це пояснюється тим, що існуючі додатки враховують швидкість руху та будують оптимальний маршрут по часу.

При порівнянні умовного «часу» маршруту, розроблений навігаційний додаток показав найкращій результат.

ВИСНОВКИ

У даній магістерській дисертації проведено дослідження існуючих засобів та технологій побудови навігаційних маршрутів. Даний аналіз показав, що наявні способи реалізації відповідного функціоналу в існуючих програмних засобах передбачають при побудові маршруту можливість обрати на власний розсуд, як критерій оптимальності, лише найкоротшу відстань або найменший час.

Значним недоліком таких мобільних навігаторів є обмеженість вибору критеріїв оптимізації при побудові маршруту. Наприклад, відсутня можливість вибору маршруту з урахуванням якості дорожнього покриття, завантаженості доріг, якості їх освітленості, інформації про аварійні ситуації, ремонтні роботи, пробки, закриття дороги або зміну напрямку руху, допустимого відхилення від найменшого часу або найкоротшої дистанції тощо.

З метою розширення функціональності таких навігаторів в частині забезпечення можливості урахування при побудові маршрутів декількох критеріїв оптимальності, були досліджені наявні методи багатокритеріальної оптимізації та алгоритми пошуку найкоротшого маршруту.

Показано, що для вирішення поставленої задачі доцільним є використання методів багатокритеріальної оптимізації, заснованих на згортанні критеріїв.

Запропоновано метод побудови навігаційних маршрутів з урахуванням декількох критеріїв оптимальності, який дозволяє отримати як оптимальний маршрут, так і множину субоптимальних рішень, відхилення яких від оптимального знаходиться в межах допустимого діапазону.

В основі даного методу лежить використання удосконаленого алгоритму пошуку, який базується на реалізації алгоритму пошуку A*. Ідея

удосконаленого алгоритму пошуку полягає у тому, що на кожному кроці його роботи відбувається не вибір зменшеної мітки, як у оригінальному алгоритмі, а додавання, отриманої на даній ітерації, мітки до множини вже існуючих. Результатом роботи такого алгоритму в загальному випадку є множина маршрутів, які представляють собою ефективні рішення даної задачі пошуку найкоротших шляхів з урахуванням декількох критеріїв оптимальності.

Використання такого алгоритму дозволяє визначити декілька субоптимальних рішень, відхилення яких від оптимального знаходиться в межах допустимого діапазону

З використанням запропонованого алгоритму розроблено мобільний навігаційний додаток, який демонструє роботу алгоритму. За додатковий критерій для побудови маршруту обрано якість дорожнього покриття. Дані про дороги оновлюються в режимі реального часу з серверу сервісу УкрЯма.

Додаток потребує допрацювань, а саме можливість пошуку об'єктів на карті. Також в подальшому плануються розширювати список додаткових критеріїв для побудови багатокритеріальних навігаційних маршрутів.

СПИСОК ЛІТЕРАТУРИ

1) GPS. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/GPS

2) Задача комівояжера. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%97%D0%B0%D0%B4%D0%B0%D1%87%D0%B0_%D0%BA%D0%BE%D0%BC%D1%96%D0%B2%D0%BE%D1%8F%D0%B6%D0%B5%D1%80%D0%B0

3) Navitel[Електронний ресурс] – Режим доступу:http://navitel.ru

4) Garmin[Електронний ресурс] – Режим доступу:http://garmin.com.ua

5) Левітін А. Алгоритми. Введення в розробку і аналіз. Вільямс, 2006. 35-36с.

6) Метод гілок і меж. [Електронний ресурс] – Режим доступу: https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4_%D0%B3%D1%96%D0%BB%D0%BE%D0%BA_%D1%96_%D0%BC%D0%B5%D0%B6

7) Розділ 1 Алгоритми пошуку найкоротших шляхів. [Електронний ресурс] – Режим доступу: https://studfiles.net/preview/4512987/page:2/

8) Алгоритм пошуку A*. [Електронний ресурс] – Режим доступу:https://uk.wikipedia.org/wiki/%D0%90%D0%BB%D0%B3%D0%BE%D1%80%D0%B8%D1%82%D0%BC_%D0%BF%D0%BE%D1%88%D1%83%D0%BA%D1%83_A*

9) Ю. В. Чибісов, Ю. С. Шульга. Застосування методів багатокритеріальної оптимізації для вирішення задачі розподілу вагонів по вантажним фронтам. //«Транспортні системи та технології перевезень». Збірник наукових праць ДНУЗТ ім. акад. В. Лазаряна. - Вип. 7. - 2014. - С. 65-72.

10) Постановка задачи векторной оптимизации [Електронний ресурс]. – Режим доступу:http://edu.nstu.ru/courses/mo_tpr/files/5.html

11) Багатокритеріальна оптимізація. Принцип Беллмана Заде. [Електронний ресурс] – Режим доступу:http://um.com.ua/8/8-2/8-232909.html

12) Розв’язування і біометричних ігор. Рівноваги Неша і Парето. [Електронний ресурс] – Режим доступу:https://infopedia.su/5x21d5.html

13) Методи розв’язання задач багатокритеріальної оптимізації. [Електронний ресурс] – Режим доступу:https://uk.wikipedia.org/wiki/%D0%9C%D0%B5%D1%82%D0%BE%D0%B4%D0%B8_%D1%80%D0%BE%D0%B7%D0%B2%27%D1%8F%D0%B7%D0%BA%D1%83_%D0%B7%D0%B0%D0%B4%D0%B0%D1%87_%D0%B1%D0%B0%D0%B3%D0%B0%D1%82%D0%BE%D0%BA%D1%80%D0%B8%D1%82%D0%B5%D1%80%D1%96%D0%B0%D0%BB%D1%8C%D0%BD%D0%BE%D1%97_%D0%BE%D0%BF%D1%82%D0%B8%D0%BC%D1%96%D0%B7%D0%B0%D1%86%D1%96%D1%

14) Багаокритеріальні задачі. [Електронний ресурс] – Режим доступу: http://studfile.net/preview/5403385/page:22

15) TheA* Algorithm. [Електронний ресурс] – Режим доступу:http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html#algorithms

16) OpenStreetMap. [Електронний ресурс] – Режим доступу:https://openstreetmaps.org.ua

17) Клієнт-серверна архітектура. [Електронний ресурс] – Режим доступу:https://uk.wikipedia.org/wiki/%D0%9A%D0%BB%D1%96%D1%94%D0%BD%D1%82-%D1%81%D0%B5%D1%80%D0%B2%D0%B5%D1%80%D0%BD%D0%B0_%D0%B0%D1%80%D1%85%D1%96%D1%82%D0%B5%D0%BA%D1%82%D1%83%D1%80%D0%B097

18) Swift. [Електронний ресурс] – Режим доступу:https://ru.wikipedia.org/wiki/Swift_(%D1%8F%D0%B7%D1%8B%D0%BA_%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D1%8F)

19) MVC. [Електронний ресурс] – Режим доступу:https://ru.wikipedia.org/wiki/Model-View-Controller

20) iOS. [Електронний ресурс] – Режим доступу:https://ru.wikipedia.org/wiki/IOS

ДОДАТОК А ДІАГРАМА КЛАСІВ

ДОДАТОК Б СХЕМА ЕКРАНІВ

ДОДАТОК В

ДОДАТОК Г ВИХІДНИЙ КОД ДОДАТКУ

```
import UIKit;
import GoogleMaps;

let GOOGLE_MAPS_SDK_API_KEY : String = "AIzaSyD80oJhn54pgyraMJfKyH0LopEakMMA9Kc";

@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

   var window: UIWindow?


   func application(_ application: UIApplication, didFinishLaunchingWithOptions launchOptions: [UIApplicationLaunchOptionsKey: Any]?)
-> Bool {
      GMSServices.provideAPIKey(GOOGLE_MAPS_SDK_API_KEY);
      return true
   }


   func applicationWillResignActive(_ application: UIApplication) {}

   func applicationDidEnterBackground(_ application: UIApplication) {}

   func applicationWillEnterForeground(_ application: UIApplication) {}

   func applicationDidBecomeActive(_ application: UIApplication) {}

   func applicationWillTerminate(_ application: UIApplication) {}


}



import UIKit
import Contacts


extension UIView {

   @IBInspectable var shadow: Bool {
      get {
         return layer.shadowOpacity > 0.0
      }
      set {
         if newValue == true {
            self.addShadow()
         }
      }
   }

   @IBInspectable var cornerRadius: CGFloat {
      get {
         return self.layer.cornerRadius
      }
      set {
         self.layer.cornerRadius = newValue

         // Don't touch the masksToBound property if a shadow is needed in addition to the cornerRadius
         if shadow == false {
            self.layer.masksToBounds = true
         }
      }
   }


   func addShadow(shadowColor: CGColor = UIColor.black.cgColor,
                  shadowOffset: CGSize = CGSize(width: 1.0, height: 2.0),
                  shadowOpacity: Float = 0.4,
                  shadowRadius: CGFloat = 3.0) {
```

```
        layer.shadowColor = shadowColor
        layer.shadowOffset = shadowOffset
        layer.shadowOpacity = shadowOpacity
        layer.shadowRadius = shadowRadius
    }
}

extension String {

    var digits: String {
        return components(separatedBy: CharacterSet.decimalDigits.inverted)
            .joined(separator: "");
    }

    var length: Int {
        return characters.count;
    }

    func image(imageSize : CGSize, textSize : CGFloat) -> UIImage {
        UIGraphicsBeginImageContextWithOptions(imageSize, false, 0);
        UIColor.clear.set();
        let rect = CGRect(origin: .zero, size: imageSize);
        UIRectFill(CGRect(origin: .zero, size: imageSize));
        self.draw(in: rect, withAttributes: [NSFontAttributeName: UIFont.systemFont(ofSize: textSize)]);
        let image = UIGraphicsGetImageFromCurrentImageContext();
        UIGraphicsEndImageContext();
        return image!
    }

    func subString(startIndex: Int, length: Int) -> String {
        let start = self.index(self.startIndex, offsetBy: startIndex);
        let end = self.index(self.startIndex, offsetBy: (startIndex + length));
        return self[start ..< end];
    }

    func subString(startIndex: Int) -> String {
        if (self.length > startIndex) {
            let start = self.index(self.startIndex, offsetBy: startIndex);
            return self[start ..< self.endIndex];
        }
        return "";
    }

}

extension URLSession {

    func synchronousDataTask(with request: URLRequest) -> Data! {

        let semaphore = DispatchSemaphore(value: 0);

        var responseData: Data?;

        URLSession.shared.dataTask(with: request) { (data, response, error) -> Void in

            responseData = data;

            semaphore.signal();

            }.resume();

        _ = semaphore.wait(timeout: .distantFuture);

        return responseData;

    }

}


let NOTIFICATION_TOKEN_ERROR : String = "NOTIFICATION_TOKEN_ERROR";
```

```
class Utils {

    class func widthForLabel(text:String, font : UIFont) -> CGFloat {
        let label:UILabel = UILabel(frame: CGRect(x: 0, y: 0, width: 1000, height: 1000));
        label.numberOfLines = 0;
        label.lineBreakMode = .byWordWrapping;
        label.font = font;
        label.text = text;

        label.sizeToFit();
        return label.frame.width;
    }

    class func heightForLabel(text:String, font : UIFont, width : CGFloat) -> CGFloat {
        let label:UILabel = UILabel(frame: CGRect(x: 0, y: 0, width: width, height: 1000));
        label.numberOfLines = 0;
        label.lineBreakMode = .byWordWrapping;
        label.font = font;
        label.text = text;

        label.sizeToFit();
        return label.frame.height;
    }

    class func canCallNumber(phoneNumber: String) -> Bool {
        if let phoneCallURL = URL(string: "tel://\(phoneNumber)") {
            let application:UIApplication = UIApplication.shared;
            return application.canOpenURL(phoneCallURL);
        }
        return false;
    }

    class func callNumber(phoneNumber: String) {
        if let phoneCallURL = URL(string: "tel://\(phoneNumber)") {
            let application:UIApplication = UIApplication.shared;
            if (application.canOpenURL(phoneCallURL)) {
                application.openURL(phoneCallURL);
            }
        }
    }

    class func isIpad() -> Bool {
        return UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiom.pad;
    }

    class func addToContacts(phoneNumber : String, firstName : String, lastName : String, photo : UIImage! = nil, parentViewController :
UIViewController) -> Bool {
        let authorizationStatus = CNContactStore.authorizationStatus(for: .contacts);
        switch authorizationStatus {
        case .authorized:
            let contactsStore = CNContactStore();
            let newContact = CNMutableContact();
            newContact.givenName = firstName;
            newContact.familyName = lastName;
            newContact.note = "Added from navinx"
            if (photo != nil) {
                let data = UIImagePNGRepresentation(photo);
                newContact.imageData = data;
            }
            let phone = CNLabeledValue(label: CNLabelWork, value:CNPhoneNumber(stringValue: phoneNumber));
            newContact.phoneNumbers = [phone];

            let request = CNSaveRequest();
            request.add(newContact, toContainerWithIdentifier: nil);
            do{
                try contactsStore.execute(request);
            } catch {
                return false;
            }
        case .denied:
            let allowContactsAlert = UIAlertController(title: "Error", message: "Please allow Contacts service", preferredStyle: .alert);
            allowContactsAlert.view.tintColor = Utils.getMainColor();
```

```
        allowContactsAlert.addAction(UIAlertAction(title: "Allow", style: .default, handler: { (action) in
            UIApplication.shared.openURL(URL(string: UIApplicationOpenSettingsURLString)!);
        }));
        allowContactsAlert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
        parentViewController.present(allowContactsAlert, animated: true, completion: nil);
        return false;
    default:
        return false;
    }
    return true;
}

class func searchForContactUsingPhoneNumber(phoneNumber: String) -> String {
    func requestForAccess(completionHandler: @escaping (_ accessGranted: Bool) -> Void) {
        let authorizationStatus = CNContactStore.authorizationStatus(for: .contacts);
        switch authorizationStatus {
        case .authorized:
            completionHandler(true)

        case .denied, .notDetermined:
            CNContactStore().requestAccess(for: .contacts, completionHandler: { (access, accessError) -> Void in
                if access {
                    completionHandler(access)
                }
            })

        default:
            completionHandler(false)
        }
    }

    var name : String = "";
    requestForAccess { (accessGranted) -> Void in
        if accessGranted {
            let keys = [CNContactGivenNameKey, CNContactFamilyNameKey, CNContactPhoneNumbersKey, CNContactNicknameKey]


            let contactsStore = CNContactStore();
            do {
                try contactsStore.enumerateContacts(with: CNContactFetchRequest(keysToFetch: keys as [CNKeyDescriptor])) {
                    (contact, cursor) -> Void in
                    if (!contact.phoneNumbers.isEmpty) {
                        let phoneNumberToCompareAgainst = Utils.formatPhoneNumber(phone: phoneNumber).subString(startIndex: 3);
                        for phoneNumber in contact.phoneNumbers {
                            let phoneNumberStruct = phoneNumber.value ;
                            let phoneNumberString = phoneNumberStruct.stringValue
                            let phoneNumberToCompare = Utils.formatPhoneNumber(phone: phoneNumberString);
                            if phoneNumberToCompare.contains(phoneNumberToCompareAgainst) {
                                name = contact.nickname;
                                if (name == "") {
                                    name = contact.givenName + " " + contact.familyName;
                                }
                                break;
                            }

                        }
                    }
                }

                if name == "" {
                  //  print("No contacts were found matching the given phone number.");
                }
            } catch {
                // print("Unable to fetch contacts.");
            }

        }


    }

    if (name == "") {
```

```
            name = "+" + phoneNumber;
        }

        return name;
    }

    class func getBackgroundColor() -> UIColor {
        return UIColor(red: 0.12, green: 0.12, blue: 0.12, alpha: 1.0);
    }

    class func getMainColor() -> UIColor {
        return UIColor(red: 0.1, green: 0.33, blue: 0.73, alpha: 1.0);
    }

    class func getStoryboard() -> UIStoryboard {
        return UIStoryboard(name: "Main", bundle: nil);
    }

    class func postInvalidTokenNotification() {
        NotificationCenter.default.post(name: Notification.Name(NOTIFICATION_TOKEN_ERROR), object: nil);
    }

    class func getWaitViewController() -> UIViewController {
        let storyboard = Utils.getStoryboard();
        let controller = storyboard.instantiateViewController(withIdentifier: "WaitViewController");
        controller.modalPresentationStyle = .overCurrentContext;

        return controller;
    }

    class func validatePhoneNumber(phone : String) -> Bool {
        return phone.rangeOfCharacter(from: CharacterSet(charactersIn: "+0123456789").inverted) == nil;
    }

    class func formatPhoneNumber(phone : String) -> String {
        return phone.digits;
    }

    class func validateSMSCode(code : String) -> Bool {
        return (code.digits == code) && (code.length == 4);
    }

    class func showAlert(title : String, message : String, parentViewController : UIViewController) {
        let alert = UIAlertController(title: title, message: message, preferredStyle: .alert);
        alert.view.tintColor = Utils.getMainColor();
        alert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
        parentViewController.present(alert, animated: true, completion: nil);
    }

    class func removeAllUserDefaults() {
        let appDomain = Bundle.main.bundleIdentifier!;
        UserDefaults.standard.removePersistentDomain(forName: appDomain);
        UserDefaults.standard.synchronize();
    }
}

class CustomViewController : UIViewController {
    override var supportedInterfaceOrientations : UIInterfaceOrientationMask {
        return .all
    }

    override var shouldAutorotate: Bool {
        return true;
    }

}

extension UINavigationController {
    override open var supportedInterfaceOrientations : UIInterfaceOrientationMask {
        return .all
    }
```

```
    override open var shouldAutorotate: Bool {
        return true;
    }
}

import Foundation;


let UDK_CURRENT_USER_PHONE = "UDK_CURRENT_USER_PHONE";
let UDK_CURRENT_USER_TOKEN = "UDK_CURRENT_USER_TOKEN";
let UDK_CURRENT_USER_EMOJI = "UDK_CURRENT_USER_EMOJI";

let CURRENT_USER_DEFAULT_EMOJI = "  ▪ ";

class CurrentUser {

    static let sharedInstance = CurrentUser();

    private init() {
        let defaults = UserDefaults.standard;
        if let phone = defaults.string(forKey: UDK_CURRENT_USER_PHONE) {
            self.phone = phone;
        } else {
            self.phone = "";
        }

        if let token = defaults.string(forKey: UDK_CURRENT_USER_TOKEN) {
            self.token = token;
        } else {
            self.token = "";
        }

        if let emoji = defaults.string(forKey: UDK_CURRENT_USER_EMOJI) {
            self.emoji = emoji;
        } else {
            self.emoji = CURRENT_USER_DEFAULT_EMOJI;
        }

    }

    func reset() {
        self.setToken(token: "");
        self.setPhone(phone: "");
        self.setEmoji(emoji: CURRENT_USER_DEFAULT_EMOJI);
    }

    func setPhone(phone : String) {
        self.phone = phone;
        UserDefaults.standard.set(self.phone, forKey: UDK_CURRENT_USER_PHONE);
        UserDefaults.standard.synchronize();
    }

    func setToken(token : String) {
        self.token = token;
        UserDefaults.standard.set(self.token, forKey: UDK_CURRENT_USER_TOKEN);
        UserDefaults.standard.synchronize();
    }

    func setEmoji(emoji : String) {
        self.emoji = emoji;
        UserDefaults.standard.set(self.emoji, forKey: UDK_CURRENT_USER_EMOJI);
        UserDefaults.standard.synchronize();
    }

    private(set) var phone : String;
    private(set) var token : String;
    private(set) var emoji : String;
    var lat : Double = 0;
    var lon : Double = 0;

}
```

```
import UIKit

class LodingViewController : CustomViewController {

   override func viewDidLoad() {
      super.viewDidLoad();
      NotificationCenter.default.addObserver(self, selector: #selector(LodingViewController.invalidToken), name:
Notification.Name(rawValue: NOTIFICATION_TOKEN_ERROR), object: nil);
   }

   var controller : UIViewController!;

   override func viewDidAppear(_ animated: Bool) {
      let currentUser = CurrentUser.sharedInstance;

      let storyboard = Utils.getStoryboard();

      if (currentUser.token == "") {
         self.controller = storyboard.instantiateViewController(withIdentifier: "LoginViewController");
         self.present(controller, animated: false, completion: nil);
      } else {
         self.controller = storyboard.instantiateViewController(withIdentifier: "MainNavigationViewController") as!
UINavigationController;
         self.present(controller, animated: false, completion: nil);
      }
   }

   func invalidToken() {
      CurrentUser.sharedInstance.reset();
      Utils.removeAllUserDefaults();
      self.controller.dismiss(animated: false, completion: nil);
   }


}


import UIKit

class LoginViewController: CustomViewController, SelectCountryDelegate {

   @IBOutlet weak var centerMainBlockConstraint: NSLayoutConstraint!;
   @IBOutlet weak var phoneNumber: UITextField!;
   @IBOutlet weak var selectCountryButton: UIButton!;

   override func viewDidLoad() {
      super.viewDidLoad();
      NotificationCenter.default.addObserver(self, selector: #selector(LoginViewController.keyboardWillShow(_:)), name:
NSNotification.Name.UIKeyboardWillShow, object: nil);
      NotificationCenter.default.addObserver(self, selector: #selector(LoginViewController.keyboardWillHide(_:)), name:
NSNotification.Name.UIKeyboardWillHide, object: nil);

      self.selectCountryButton.titleLabel?.adjustsFontSizeToFitWidth = true;
      self.selectCountryButton.titleLabel?.minimumScaleFactor = 0.5;
   }

   override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
      if (segue.identifier == "selectCountry") {
         ((segue.destination as! UINavigationController).viewControllers[0] as! SelectCountryViewController).delegate = self;
      }
   }


   @IBAction func onBackground(_ sender: Any) {
      self.view.endEditing(true);
   }

   @IBAction func onLoginButton(_ sender: Any) {
      self.view.endEditing(true);
      let phone = Utils.formatPhoneNumber(phone: self.selectCountryButton.titleLabel!.text! + self.phoneNumber.text!);
      if (Utils.validatePhoneNumber(phone: "+\(phone)")) {
```

```
        CurrentUser.sharedInstance.setPhone(phone: phone);

        let controller = Utils.getWaitViewController();
        self.present(controller, animated: false, completion: nil);

        DispatchQueue.global().async {

            let r = ServerDataLoader.registration(phone: phone);
            DispatchQueue.main.async {
                controller.dismiss(animated: false, completion: {
                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r.error == REQUST_ERROR_OK) {
                        let enterSMSCodeAlert = UIAlertController(title: "Enter code from SMS", message: "", preferredStyle: .alert);
                        enterSMSCodeAlert.view.tintColor = Utils.getMainColor();
                        enterSMSCodeAlert.addAction(UIAlertAction(title: "OK", style: .default, handler: { (action) in
                            let code = enterSMSCodeAlert.textFields![0].text!;
                            if (Utils.validateSMSCode(code: code)) {
                                let controller = Utils.getWaitViewController();
                                self.present(controller, animated: false, completion: nil);

                                DispatchQueue.global().async {

                                    let r = ServerDataLoader.checkCode(phone: CurrentUser.sharedInstance.phone, code: code);
                                    DispatchQueue.main.async {
                                        controller.dismiss(animated: false, completion: {
                                            if (r.error == REQUST_ERROR_LOAD_DATA) {
                                                Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController:
self);
                                            }
                                            if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                                                Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                                            }
                                            if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                                                Utils.postInvalidTokenNotification();
                                            }

                                            if (r.error == REQUST_ERROR_INVALID_SMS_CODE) {
                                                Utils.showAlert(title: "Invalid code from SMS", message: "Try again", parentViewController: self);
                                            }
                                            if (r.error == REQUST_ERROR_OK) {
                                                CurrentUser.sharedInstance.setToken(token: r.token);
                                                CurrentUser.sharedInstance.setEmoji(emoji: r.emoji != "" ? r.emoji : CURRENT_USER_DEFAULT_EMOJI);

                                                self.dismiss(animated: false, completion: nil);
                                            }

                                        });

                                    }
                                }
                            } else {
                                Utils.showAlert(title: "Invalid code from SMS", message: "Try again", parentViewController: self);
                            }
                        }));

                        enterSMSCodeAlert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
                        enterSMSCodeAlert.addTextField { (textField) in
                            textField.placeholder = "Code from SMS";
                            textField.keyboardType = .numberPad;
                            textField.autocorrectionType = .no;
                            textField.autocapitalizationType = .none;
                            textField.keyboardAppearance = .dark;
                        }
                        self.present(enterSMSCodeAlert, animated: true, completion: nil);
```

```
                }

            });

        }
      }
    } else {
      Utils.showAlert(title: "Invalid phone number", message: "Try again", parentViewController: self);
    }
  }

  func keyboardWillShow(_ notification: Notification) {
    let userInfo = (notification as NSNotification).userInfo!;

    let animationDuration = userInfo[UIKeyboardAnimationDurationUserInfoKey] as! Double;
    let keyboardEndFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as! NSValue).cgRectValue;

    let d = ((self.view.frame.height - (self.phoneNumber.superview!.convert(self.phoneNumber.frame, to: self.view).origin.y +
self.phoneNumber.frame.height)) - keyboardEndFrame.height);
    if (d < 50) {
      centerMainBlockConstraint.constant = d - 50;
      UIView.animate(withDuration: animationDuration, animations: { () -> Void in
        self.view.layoutIfNeeded();
      })
    }


  }

  func keyboardWillHide(_ notification: Notification) {
    let userInfo = (notification as NSNotification).userInfo!;

    let animationDuration = userInfo[UIKeyboardAnimationDurationUserInfoKey] as! Double;

    self.centerMainBlockConstraint.constant = 0.0;
    UIView.animate(withDuration: animationDuration, animations: { () -> Void in
      self.view.layoutIfNeeded();
    })

  }

  func countrySelected(country: Country) {
    self.selectCountryButton.setTitle("\(country.flag) +\(country.code)", for: .normal);
    self.phoneNumber.text = "";
  }


}



import UIKit

class GroupsViewController : CustomViewController, UITableViewDelegate, UITableViewDataSource, UISearchResultsUpdating,
UISearchControllerDelegate {

  @IBOutlet weak var tableView: UITableView!;
  @IBOutlet weak var phoneNumber: UILabel!;
  @IBOutlet weak var settingsButton: UIButton!;

  let searchController = UISearchController(searchResultsController: nil);

  var groups : [String] = [];

  lazy var refreshControll : UIRefreshControl = {
    let refreshControl = UIRefreshControl();
    refreshControl.addTarget(self, action: #selector(GroupsViewController.updateGroups), for: .valueChanged);
    refreshControl.tintColor = .white;
    return refreshControl;
  }();

  override func viewDidLoad() {
```

```
        super.viewDidLoad();

        self.tableView.delegate = self;
        self.tableView.dataSource = self;
        self.tableView.tableFooterView = UIView();
        self.tableView.backgroundView = UIView();

        self.phoneNumber.text = "+" + CurrentUser.sharedInstance.phone;

        self.searchController.searchBar.barTintColor = self.tableView.backgroundColor;
        self.searchController.searchBar.returnKeyType = .default;
        self.searchController.view.tintColor = Utils.getMainColor();
        self.searchController.searchBar.keyboardAppearance = .dark;
        (self.searchController.searchBar.value(forKey: "searchField") as? UITextField)?.textColor = Utils.getMainColor();
        self.searchController.searchResultsUpdater = self;
        self.searchController.delegate = self;
        self.searchController.dimsBackgroundDuringPresentation = false;
        self.definesPresentationContext = true;
        self.tableView.tableHeaderView = searchController.searchBar;

        self.tableView.insertSubview(refreshControll, at: 0);

        self.updateGroups();
    }

    override func viewWillAppear(_ animated: Bool) {
        super.viewWillAppear(animated);
        self.settingsButton.setImage(CurrentUser.sharedInstance.emoji.image(imageSize: CGSize(width: 30, height: 35), textSize: 30), for:
.normal);
    }

    var updating : Bool = false;
    func updateGroups() {
        if (self.searchController.isActive) {
            self.refreshControll.endRefreshing();
            return;
        }
        if (!updating) {
            self.updating = true;
            DispatchQueue.global().async {
                let r = ServerDataLoader.getGroups(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token);

                DispatchQueue.main.async {
                    self.refreshControll.endRefreshing();
                    self.updating = false;
                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r.error == REQUST_ERROR_OK) {
                        self.refreshControll.endRefreshing();
                        self.groups = r.groups;
                        if (!self.searchController.isActive) {
                            self.tableView.reloadData();
                        }

                    }


                }
            }
        }


    }
```

```
    @IBAction func onAddGroup(_ sender: Any) {

        let addGroupAlert = UIAlertController(title: "Connect to group", message: "or create group", preferredStyle: .alert);
        addGroupAlert.view.tintColor = Utils.getMainColor();
        addGroupAlert.addAction(UIAlertAction(title: "Connect", style: .default, handler: { (action) in

            let groupName = addGroupAlert.textFields![0].text!;
            let groupPassword = addGroupAlert.textFields![1].text!;


            if (self.groups.contains(groupName)) {
                Utils.showAlert(title: "This group already added", message: "", parentViewController: self);
                return;
            }

            if (groupName.length < 3) {
                Utils.showAlert(title: "Group name must contain more 3 symbols", message: "Try again", parentViewController: self);
                return;
            }

            if (groupPassword.length < 3) {
                Utils.showAlert(title: "Group password must contain more 3 symbols", message: "Try again", parentViewController: self);
                return;
            }

            let controller = Utils.getWaitViewController();
            self.navigationController?.present(controller, animated: false, completion: nil);

            DispatchQueue.global().async {

                let r = ServerDataLoader.addGroup(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: groupName, groupPassword: groupPassword);

                DispatchQueue.main.async {
                    controller.dismiss(animated: false, completion: nil);

                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r.error == REQUST_ERROR_INVALID_GROUP_PASSWORD) {
                        Utils.showAlert(title: "Incorrect password", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_ALREADY_CONNECT_GROUP) {
                        Utils.showAlert(title: "You are already connect to this group", message: "", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_OK) {
                        self.groups.append(groupName);
                        self.tableView.reloadData();


                    }


                }
            }
        }));
        addGroupAlert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
        addGroupAlert.addTextField { (textField) in
            textField.placeholder = "Group name";
            textField.keyboardType = .asciiCapable;
            textField.autocorrectionType = .no;
            textField.autocapitalizationType = .none;
            textField.keyboardAppearance = .dark;
        }

        addGroupAlert.addTextField { (textField) in
```

```
            textField.placeholder = "Group password";
            textField.keyboardType = .asciiCapable;
            textField.isSecureTextEntry = true;
            textField.autocorrectionType = .no;
            textField.autocapitalizationType = .none;
            textField.keyboardAppearance = .dark;
        }
        self.present(addGroupAlert, animated: true, completion: nil);
    }

    @IBAction func onLogout(_ sender: Any) {
        let logoutAlertController = UIAlertController(title: "Do you really want to logout?", message: "", preferredStyle: .alert);
        logoutAlertController.view.tintColor = Utils.getMainColor();
        logoutAlertController.addAction(UIAlertAction(title: "YES", style: .default, handler: { (action) in
            CurrentUser.sharedInstance.reset();
            Utils.removeAllUserDefaults();
            self.dismiss(animated: false, completion: nil);
        }));
        logoutAlertController.addAction(UIAlertAction(title: "NO", style: .cancel, handler: { (action) in }));
        self.present(logoutAlertController, animated: true, completion: nil);

    }


    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.groups.count;
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "GroupTableViewCell", for: indexPath) as! GroupTableViewCell;
        cell.initializate(groupName: self.groups[indexPath.item]);
        return cell;
    }


    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.cellForRow(at: indexPath)?.isSelected = false;
        self.searchController.isActive = false;

        let storyboard = Utils.getStoryboard();

        let controller = storyboard.instantiateViewController(withIdentifier: "MapViewController") as! MapViewController;
        controller.groupName = self.groups[indexPath.item];
        self.present(controller, animated: false, completion: nil);

    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        if ((self.groups[indexPath.item].lowercased().contains(self.searchController.searchBar.text!.lowercased())) ||
(self.searchController.searchBar.text! == "")) {
            return 65;
        }
        return 0;
    }

    func tableView(_ tableView: UITableView, canEditRowAt indexPath: IndexPath) -> Bool {
        return true
    }

    func tableView(_ tableView: UITableView, commit editingStyle: UITableViewCellEditingStyle, forRowAt indexPath: IndexPath) {
        if (editingStyle == .delete) {
            let controller = Utils.getWaitViewController();
            self.present(controller, animated: false, completion: nil);

            DispatchQueue.global().async {


                let r = ServerDataLoader.removeGroup(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groups[indexPath.item]);

                DispatchQueue.main.async {
                    controller.dismiss(animated: false, completion: nil);
```

```
                if (r.error == REQUST_ERROR_LOAD_DATA) {
                    Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                }
                if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                    Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                }
                if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                    Utils.postInvalidTokenNotification();
                }
                if (r.error == REQUST_ERROR_OK) {
                    self.groups.remove(at: indexPath.item);
                    self.tableView.reloadData();

                }


            }
        }

    }
  }

  func doSearch() {
    self.tableView.reloadData();
  }

  var searchTimer : Timer? = nil;
  func updateSearchResults(for searchController: UISearchController) {
    if (self.searchTimer != nil) {
      self.searchTimer?.invalidate();
      self.searchTimer = nil;
    }
    self.searchTimer = Timer.scheduledTimer(timeInterval: 0.25, target: self, selector: #selector(SelectCountryViewController.doSearch),
userInfo: nil, repeats: false);

  }

  func willPresentSearchController(_ searchController: UISearchController) {
    self.refreshControll.isHidden = true;
  }

  func didDismissSearchController(_ searchController: UISearchController) {
    self.refreshControll.isHidden = false;
  }

}


import UIKit


class SettingsViewController : CustomViewController, ISEmojiViewDelegate {



  var emojiSelectView: ISEmojiView!;
  @IBOutlet weak var emoji: UILabel!;


  override func viewDidLoad() {
    super.viewDidLoad();
    self.emoji.text = " " + CurrentUser.sharedInstance.emoji;
    self.redrawEmoji();
    NotificationCenter.default.addObserver(self, selector: #selector(SettingsViewController.redrawEmoji), name:
NSNotification.Name.UIDeviceOrientationDidChange, object: nil);
  }

  func redrawEmoji() {
    var w = self.view.frame.size.width;
    var h = self.view.frame.size.height;
    switch UIDevice.current.orientation {
```

```
        case .portrait, .portraitUpsideDown, .faceUp, .faceDown:
            if (w > h) {
                let t = w;
                w = h;
                h = t;
            }
        default:
            if (w < h) {
                let t = w;
                w = h;
                h = t;
            }

        }
        if (self.emojiSelectView != nil) {
            self.emojiSelectView.removeFromSuperview();
            self.emojiSelectView = nil;
        }
        self.emojiSelectView = ISEmojiView(frame: CGRect.init(x: 0, y: h * 0.5, width: w, height: h * 0.5));
        self.emojiSelectView.delegate = self;
        self.view.addSubview(self.emojiSelectView);


    }

    @IBAction func onSave(_ sender: Any) {
        let controller = Utils.getWaitViewController();
        self.present(controller, animated: false, completion: nil);

        DispatchQueue.global().async {

            let r = ServerDataLoader.setEmoji(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token, emoji:
self.emoji!.text!.subString(startIndex: 1));

            DispatchQueue.main.async {
                controller.dismiss(animated: false, completion: {
                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r.error == REQUST_ERROR_OK) {
                        CurrentUser.sharedInstance.setEmoji(emoji: self.emoji!.text!.subString(startIndex: 1));
                        self.navigationController!.popViewController(animated: true);
                    }
                })




            }
        }

    }

    func emojiViewDidSelectEmoji(emojiView: ISEmojiView, emoji: String) {
        self.emoji.text = " " + emoji;
    }

}


import UIKit
import GoogleMaps

let STATIC_POINT_EMOJI : String = "  ";
let MEETING_POINT_EMOJI : String = "  ";
```

```
class MapViewController : CustomViewController, GMSMapViewDelegate, CLLocationManagerDelegate {

    var groupName : String!;
    @IBOutlet weak var map: GMSMapView!;
    let locationManager = CLLocationManager();
    var userMarker : MapMarker! = nil;
    var uMarkers : [MapMarker] = [];
    var pMarkers : [MapMarker] = [];
    var responce : GetGroupResponce = GetGroupResponce();

    var userRoute : GMSPolyline? = nil;
    var routeMarkers : [MapMarker] = [];

    var timer : Timer!;


    @IBOutlet weak var mapViewYConstraint: NSLayoutConstraint!;


    override func viewDidLoad() {
        super.viewDidLoad();

        self.mapViewYConstraint.constant = 0;

        self.locationManager.delegate = self;
        self.locationManager.startUpdatingLocation();

        self.map.delegate = self;
        self.map.settings.compassButton = true;

        self.timer = Timer.scheduledTimer(timeInterval: 5, target: self, selector: #selector(MapViewController.timerUpdate), userInfo: nil,
repeats: true);
        self.timerUpdate();

    }

    override func viewDidAppear(_ animated: Bool) {
        super.viewDidAppear(animated);

        switch CLLocationManager.authorizationStatus() {
        case .denied:
            let allowLocationAlert = UIAlertController(title: "Update location error", message: "Please allow location service", preferredStyle:
.alert);
            allowLocationAlert.view.tintColor = Utils.getMainColor();
            allowLocationAlert.addAction(UIAlertAction(title: "Allow", style: .default, handler: { (action) in
                UIApplication.shared.openURL(URL(string: UIApplicationOpenSettingsURLString)!);
            }));
            allowLocationAlert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
            self.present(allowLocationAlert, animated: true, completion: nil);
        case .notDetermined:
            locationManager.requestWhenInUseAuthorization();
        default:
            break;
        }
    }

    var updatingGroup = false;
    var updatingCoords = false;
    func timerUpdate() {
        if (!self.updatingGroup) {
            DispatchQueue.global().async {
                self.updatingGroup = true;
                let r = ServerDataLoader.getGroup(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName);
                DispatchQueue.main.async {
                    self.updatingGroup = false;
                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
```

```
            if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                Utils.postInvalidTokenNotification();
            }
            if (r.error == REQUST_ERROR_OK) {
                self.responce = r;

                for i in 0 ..< self.responce.users.count {
                    self.responce.users[i].name = Utils.searchForContactUsingPhoneNumber(phoneNumber: self.responce.users[i].phone);
                    var fl : Bool = false;
                    for m in self.uMarkers {
                        if (m.id == "u\(r.users[i].phone)") {
                            m.position = CLLocationCoordinate2D(latitude: self.responce.users[i].lat, longitude: self.responce.users[i].lon);
                            m.icon = self.responce.users[i].emoji.image(imageSize: CGSize(width: 30, height: 35), textSize: 30);
                            m.updated = true;
                            fl = true;
                            break;
                        }
                    }
                    if (!fl) {
                        let marker = MapMarker(position: CLLocationCoordinate2D(latitude: self.responce.users[i].lat, longitude:
self.responce.users[i].lon));
                        marker?.title = self.responce.users[i].name;
                        marker?.icon = self.responce.users[i].emoji.image(imageSize: CGSize(width: 30, height: 35), textSize: 30);
                        marker?.groundAnchor = CGPoint(x: 0.5, y: 0.5);
                        marker?.map = self.map;
                        marker?.id = "u\(r.users[i].phone)";
                        marker?.updated = true;
                        marker?.phone = self.responce.users[i].phone;
                        self.uMarkers.append(marker!);
                    }

                }

                for m in self.uMarkers {
                    if (!m.updated) {
                        m.map = nil;
                        self.uMarkers.remove(at: self.uMarkers.index(of: m)!);
                    } else {
                        m.updated = false;
                    }
                }

                for i in 0 ..< self.responce.offlineUsers.count {
                    self.responce.offlineUsers[i].name = Utils.searchForContactUsingPhoneNumber(phoneNumber:
self.responce.offlineUsers[i].phone);
                }


                for i in 0 ..< r.points.count {
                    var fl : Bool = false;
                    for m in self.pMarkers {
                        if (m.id == "p\(r.points[i].name)\(r.points[i].lat)\(r.points[i].lon)") {
                            m.updated = true;
                            fl = true;
                            break;
                        }
                    }
                    if (!fl) {
                        let marker = MapMarker(position: CLLocationCoordinate2D(latitude: r.points[i].lat, longitude:  r.points[i].lon));
                        marker?.title =  r.points[i].name;
                        marker?.isUser = false;
                        marker?.icon = STATIC_POINT_EMOJI.image(imageSize: CGSize(width: 15, height: 15), textSize: 12);
                        marker?.groundAnchor = CGPoint(x: 0.5, y: 0.5);
                        marker?.map = self.map;
                        marker?.id = "p\(r.points[i].name)\(r.points[i].lat)\(r.points[i].lon)";
                        marker?.updated = true;
                        self.pMarkers.append(marker!);
                    }

                }

                for m in self.pMarkers {
```

```
                    if (!m.updated) {
                        m.map = nil;
                        self.pMarkers.remove(at: self.pMarkers.index(of: m)!);
                    } else {
                        m.updated = false;
                    }
                }

                if ((self.map.selectedMarker != nil) && (self.map.selectedMarker != self.userMarker) && ((self.map.selectedMarker as!
MapMarker).isUser)) {
                    self.map.animate(toLocation: self.map.selectedMarker.position);
                }

            }
        }
    }


    if ((self.locationManager.location != nil) && (!self.updatingCoords)){
        DispatchQueue.global().async {
            self.updatingCoords = true;
            let lat : Double = (self.locationManager.location?.coordinate.latitude)!;
            let lon : Double = (self.locationManager.location?.coordinate.longitude)!;

            let r = ServerDataLoader.updateCoords(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName, lat: lat, lon: lon);
            DispatchQueue.main.async {
                self.updatingCoords = false;
                if (r.error == REQUST_ERROR_LOAD_DATA) {
                    Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                }
                if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                    Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                }
                if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                    Utils.postInvalidTokenNotification();
                }
            }
        }
    }

}

@IBAction func onCloseRoute(_ sender: Any) {
    if (self.routeTimer != nil) {
        self.routeTimer.invalidate();
        self.routeTimer = nil;
    }
    self.closeRoute();
}

private func closeRoute(hideRouteView : Bool = true) {
    if (self.userRoute != nil) {
        self.userRoute?.map = nil;
    }

    for r in self.routeMarkers {
        r.map = nil;
    }
    self.routeMarkers.removeAll();

    if (hideRouteView) {
        UIView.animate(withDuration: 0.5, animations: {
            self.mapViewYConstraint.constant = 0;
        });
    }
}


@IBAction func onBack(_ sender: Any) {
    let logoutAlertController = UIAlertController(title: "Do you really want to exit?", message: "", preferredStyle: .alert);
```

```
        logoutAlertController.view.tintColor = Utils.getMainColor();
        logoutAlertController.addAction(UIAlertAction(title: "YES", style: .default, handler: { (action) in
            if (self.timer != nil) {
                self.timer!.invalidate();
            }
            self.dismiss(animated: false, completion: nil);
        }));
        logoutAlertController.addAction(UIAlertAction(title: "NO", style: .cancel, handler: { (action) in }));
        self.present(logoutAlertController, animated: true, completion: nil);
    }

    @IBAction func onPeople(_ sender: Any) {
        let alert = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet);
        alert.view.tintColor = Utils.getMainColor();
        for u in self.responce.users {
            alert.addAction(UIAlertAction(title: u.emoji + " " + u.name, style: .default, handler: { (action) in
                self.map.selectedMarker = nil;
                self.map.animate(toLocation: CLLocationCoordinate2D(latitude: u.lat, longitude: u.lon));
                self.map.animate(toZoom: 18);
            }));
        }
        for i in 0 ..< self.responce.offlineUsers.count {
            let u = self.responce.offlineUsers[i];
            alert.addAction(UIAlertAction(title: "offline " + u.emoji + " " + u.name, style: .default, handler: { (action) in
                var fl = false;

                let askActionController = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet);
                askActionController.view.tintColor = Utils.getMainColor();

                if ("+\(u.phone)" == u.name) {
                    fl = true;
                    askActionController.addAction(UIAlertAction(title: "Add to Contacts", style: .default, handler: { (action) in

                        let addAlertController = UIAlertController(title: "Add +\(u.phone) to contacts?", message: "", preferredStyle: .alert);
                        addAlertController.view.tintColor = Utils.getMainColor();

                        addAlertController.addTextField { (textField) in
                            textField.placeholder = "First Name";
                            textField.keyboardType = .default;
                            textField.autocorrectionType = .no;
                            textField.autocapitalizationType = .none;
                            textField.keyboardAppearance = .dark;
                        }

                        addAlertController.addTextField { (textField) in
                            textField.placeholder = "Last Name";
                            textField.keyboardType = .default;
                            textField.autocorrectionType = .no;
                            textField.autocapitalizationType = .none;
                            textField.keyboardAppearance = .dark;
                        }


                        addAlertController.addAction(UIAlertAction(title: "Add", style: .default, handler: { (action) in
                            let fn = addAlertController.textFields![0].text!;
                            let ln = addAlertController.textFields![1].text!;
                            if (Utils.addToContacts(phoneNumber: "+\(u.phone)", firstName: fn, lastName: ln, photo: u.emoji.image(imageSize:
CGSize(width: 20, height: 25), textSize: 17), parentViewController: self)) {
                                self.responce.offlineUsers[i].name = "\(fn) \(ln)";
                            }

                        }));
                        addAlertController.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
                        self.present(addAlertController, animated: true, completion: nil);


                    }));
                }
                if (Utils.canCallNumber(phoneNumber: "+" + u.phone)) {
                    fl = true;
                    askActionController.addAction(UIAlertAction(title: "Call \(u.name)", style: .default, handler: { (action) in
                        let callAlertController = UIAlertController(title: "Call \(u.name)?", message: "", preferredStyle: .alert);
```

```
                callAlertController.view.tintColor = Utils.getMainColor();


                callAlertController.addAction(UIAlertAction(title: "YES", style: .default, handler: { (action) in
                    Utils.callNumber(phoneNumber: "+" + u.phone);
                }));


                callAlertController.addAction(UIAlertAction(title: "NO", style: .cancel, handler: { (action) in }));
                self.present(callAlertController, animated: true, completion: nil);
            }));
        }
        if (fl) {
            askActionController.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
            askActionController.popoverPresentationController?.sourceView = self.view;
            askActionController.popoverPresentationController?.sourceRect = self.view.bounds;

            if (Utils.isIpad()){
                let s = sender as! UIView;
                let x = s.bounds.midX;
                let y = s.bounds.minY;
                askActionController.modalPresentationStyle = .popover;
                askActionController.popoverPresentationController?.sourceView = s;
                askActionController.popoverPresentationController?.sourceRect = CGRect(x: x, y: y, width: 0, height: 0);

            }
            self.present(askActionController, animated: true, completion: nil);
        }
    }));
  }

  alert.addAction(UIAlertAction(title: "Close", style: .cancel, handler: { (action) in
  }));

  if (Utils.isIpad()){
      let s = sender as! UIView;
      let x = s.bounds.midX;
      let y = s.bounds.minY;
      alert.modalPresentationStyle = .popover;
      alert.popoverPresentationController?.sourceView = s;
      alert.popoverPresentationController?.sourceRect = CGRect(x: x, y: y, width: 0, height: 0);

  }

  self.present(alert, animated: true, completion: nil);




}

@IBAction func onCenter(_ sender: Any) {
  if let l = self.locationManager.location {
      self.map.animate(toLocation: l.coordinate);
      self.map.animate(toZoom: 18);
      self.map.selectedMarker = self.userMarker;
  }

}

@IBAction func onPoints(_ sender: Any) {
  let alert = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet);
  alert.view.tintColor = Utils.getMainColor();
  for p in self.responce.points {
      alert.addAction(UIAlertAction(title: STATIC_POINT_EMOJI + " " + p.name, style: .default, handler: { (action) in
          self.map.selectedMarker = nil;
          self.map.animate(toLocation: CLLocationCoordinate2D(latitude: p.lat, longitude: p.lon));
          self.map.animate(toZoom: 18);
      }));
  }
  alert.addAction(UIAlertAction(title: "Close", style: .cancel, handler: { (action) in
  }));
```

```
        alert.popoverPresentationController?.sourceView = self.view;
        alert.popoverPresentationController?.sourceRect = self.view.bounds;

        if (Utils.isIpad()){
            let s = sender as! UIView;
            let x = s.bounds.midX;
            let y = s.bounds.minY;
            alert.modalPresentationStyle = .popover;
            alert.popoverPresentationController?.sourceView = s;
            alert.popoverPresentationController?.sourceRect = CGRect(x: x, y: y, width: 0, height: 0);

        }

        self.present(alert, animated: true, completion: nil);

    }

    @IBAction func onMessages(_ sender: Any) {
        //TODO
    }


    func mapView(_ mapView: GMSMapView!, didLongPressAt coordinate: CLLocationCoordinate2D) {

        let addPointAlert = UIAlertController(title: "Enter point name", message: "", preferredStyle: .alert);
        addPointAlert.view.tintColor = Utils.getMainColor();
        addPointAlert.addAction(UIAlertAction(title: "OK", style: .default, handler: { (action) in
            let pointName = addPointAlert.textFields![0].text!;

            if (pointName.length < 3) {
                Utils.showAlert(title: "Point name must contain more 3 symbols", message: "Try again", parentViewController: self);
                return;
            }

            let controller = Utils.getWaitViewController();
            self.present(controller, animated: false, completion: nil);
            DispatchQueue.global().async {


                let r = ServerDataLoader.addPoint(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
    groupName: self.groupName, pointName: pointName, lat: coordinate.latitude, lon: coordinate.longitude);
                DispatchQueue.main.async {
                    controller.dismiss(animated: false, completion: {
                        if (r.error == REQUST_ERROR_LOAD_DATA) {
                            Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                        }
                        if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                            Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                        }
                        if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                            Utils.postInvalidTokenNotification();
                        }
                        if (r.error == REQUST_ERROR_OK) {
                            let marker = MapMarker(position: coordinate);
                            marker?.title = pointName;
                            marker?.isUser = false;
                            marker?.icon = STATIC_POINT_EMOJI.image(imageSize: CGSize(width: 15, height: 15), textSize: 12);
                            marker?.groundAnchor = CGPoint(x: 0.5, y: 0.5);
                            marker?.map = self.map;
                            marker?.id = "p\(pointName)\(coordinate.latitude)\(coordinate.longitude)"
                            self.pMarkers.append(marker!);
                        }
                    })

                }
            }



        }));

        addPointAlert.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
```

```
        addPointAlert.addTextField { (textField) in
            textField.placeholder = "Point name";
            textField.autocorrectionType = .no;
            textField.autocapitalizationType = .none;
            textField.keyboardAppearance = .dark;
            textField.keyboardType = .asciiCapable;
        }
        self.present(addPointAlert, animated: true, completion: nil);
    }


    func mapView(_ mapView: GMSMapView!, didTapInfoWindowOf marker: GMSMarker!) {
        let m = marker as! MapMarker;
        if (!m.isUser) {
            if (m.phone == "MP") {
                return;//todo
            }

            let askActionController = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet);
            askActionController.view.tintColor = Utils.getMainColor();

            askActionController.addAction(UIAlertAction(title: "Delete point \"\(m.title!)\"", style: .default, handler: { (action) in
                let deletePointAlertController = UIAlertController(title: "Do you really want to delete point?", message: "", preferredStyle: .alert);
                deletePointAlertController.view.tintColor = Utils.getMainColor();
                deletePointAlertController.addAction(UIAlertAction(title: "YES", style: .default, handler: { (action) in

                    let controller = Utils.getWaitViewController();
                    self.present(controller, animated: false, completion: nil);
                    DispatchQueue.global().async {


                        let r = ServerDataLoader.removePoint(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName, pointName: marker.title, lat: marker.position.latitude, lon: marker.position.longitude);
                        DispatchQueue.main.async {
                            controller.dismiss(animated: false, completion: {
                                if (r.error == REQUST_ERROR_LOAD_DATA) {
                                    Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                                }
                                if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                                    Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                                }
                                if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                                    Utils.postInvalidTokenNotification();
                                }
                                if (r.error == REQUST_ERROR_OK) {
                                    marker.map = nil;
                                    self.pMarkers.remove(at: self.pMarkers.index(of: marker as! MapMarker)!);
                                }
                            })

                        }
                    }


                }));
                deletePointAlertController.addAction(UIAlertAction(title: "NO", style: .cancel, handler: { (action) in }));
                self.present(deletePointAlertController, animated: true, completion: nil);
            }));

            askActionController.addAction(UIAlertAction(title: "Route here", style: .default, handler: { (action) in
                self.map.selectedMarker = nil;
                if (self.userMarker != nil) {
                    let controller = Utils.getWaitViewController();
                    self.present(controller, animated: false, completion: nil);
                    DispatchQueue.global().async {
                        let r = ServerDataLoader.getDirection(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName : self.groupName, startLat: self.userMarker.position.latitude, startLon: self.userMarker.position.longitude, endLat:
m.position.latitude, endLon: m.position.longitude);
                        DispatchQueue.main.async {
                            controller.dismiss(animated: false, completion: {
                                if (r.error == REQUST_ERROR_LOAD_DATA) {
                                    Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
```

```
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r.error == REQUST_ERROR_OK) {
                        if (self.routeTimer != nil) {
                            self.routeTimer.invalidate();
                            self.routeTimer = nil;
                        }
                        self.endPoint = LatLon(lat: m.position.latitude, lon: m.position.longitude);
                        self.routeTimer = Timer.scheduledTimer(timeInterval: 10, target: self, selector:
#selector(MapViewController.updateRouteTimer), userInfo: nil, repeats: true);
                        self.buildRoute(r: r);

                    }
                })

            }
        }

    }

});

askActionController.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
askActionController.popoverPresentationController?.sourceView = self.view;
askActionController.popoverPresentationController?.sourceRect = self.view.bounds;

if (Utils.isIpad()){
    let s = self.map;
    let locationInView = self.map.projection.point(for: marker.position);
    let x = locationInView.x;
    let y = locationInView.y;
    askActionController.modalPresentationStyle = .popover;
    askActionController.popoverPresentationController?.sourceView = s;
    askActionController.popoverPresentationController?.sourceRect = CGRect(x: x - 20, y: y - 20, width: 40, height: 40);

}
self.present(askActionController, animated: true, completion: nil);







} else {


    if (m.phone != CurrentUser.sharedInstance.phone) {

        let askActionController = UIAlertController(title: nil, message: nil, preferredStyle: .actionSheet);
        askActionController.view.tintColor = Utils.getMainColor();

        var fl = false;
        if ("+\(m.phone)" == m.title) {
            fl = true;
            askActionController.addAction(UIAlertAction(title: "Add to Contacts", style: .default, handler: { (action) in

                let addAlertController = UIAlertController(title: "Add \(m.title!) to contacts?", message: "", preferredStyle: .alert);
                addAlertController.view.tintColor = Utils.getMainColor();

                addAlertController.addTextField { (textField) in
                    textField.placeholder = "First Name";
                    textField.keyboardType = .default;
                    textField.autocorrectionType = .no;
                    textField.autocapitalizationType = .none;
                    textField.keyboardAppearance = .dark;
```

```
            }

            addAlertController.addTextField { (textField) in
               textField.placeholder = "Last Name";
               textField.keyboardType = .default;
               textField.autocorrectionType = .no;
               textField.autocapitalizationType = .none;
               textField.keyboardAppearance = .dark;
            }


            addAlertController.addAction(UIAlertAction(title: "Add", style: .default, handler: { (action) in
               let fn = addAlertController.textFields![0].text!;
               let ln = addAlertController.textFields![1].text!;
               if (Utils.addToContacts(phoneNumber: m.title, firstName: fn, lastName: ln, photo: m.icon, parentViewController: self)) {
                  if (self.userMarker.index(ofAccessibilityElement: marker) > -1) {
                     for i in 0 ..< self.responce.users.count {
                        if ("+\(self.responce.users[i].phone)" == m.title) {
                           self.responce.users[i].name = "\(fn) \(ln)";
                           break;
                        }
                     }

                     marker.title = "\(fn) \(ln)";

                  }

               }

            }));
            addAlertController.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));

            self.present(addAlertController, animated: true, completion: nil);


         }));
      }
      if (Utils.canCallNumber(phoneNumber: "+" + m.phone)) {
         fl = true;
         askActionController.addAction(UIAlertAction(title: "Call \(m.title!)", style: .default, handler: { (action) in
            let callAlertController = UIAlertController(title: "Call \(m.title!)?", message: "", preferredStyle: .alert);
            callAlertController.view.tintColor = Utils.getMainColor();

            callAlertController.addAction(UIAlertAction(title: "YES", style: .default, handler: { (action) in
               Utils.callNumber(phoneNumber: "+" + m.phone);
            }));

            callAlertController.addAction(UIAlertAction(title: "NO", style: .cancel, handler: { (action) in }));
            self.present(callAlertController, animated: true, completion: nil);
         }));
      }
      if (fl) {
         askActionController.addAction(UIAlertAction(title: "Cancel", style: .cancel, handler: { (action) in }));
         askActionController.popoverPresentationController?.sourceView = self.view;
         askActionController.popoverPresentationController?.sourceRect = self.view.bounds;

         if (Utils.isIpad()){
            let s = self.map;
            let locationInView = self.map.projection.point(for: marker.position);
            let x = locationInView.x;
            let y = locationInView.y;
            askActionController.modalPresentationStyle = .popover;
            askActionController.popoverPresentationController?.sourceView = s;
            askActionController.popoverPresentationController?.sourceRect = CGRect(x: x - 20, y: y - 20, width: 40, height: 40);

         }
         self.present(askActionController, animated: true, completion: nil);
      }
   }

  }
}
```

```
    var routeTimer : Timer! = nil;
    var endPoint : LatLon!;
    func updateRouteTimer() {
        DispatchQueue.global().async {
            let r = ServerDataLoader.getDirection(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName : self.groupName, startLat: self.userMarker.position.latitude, startLon: self.userMarker.position.longitude, endLat:
self.endPoint.lat, endLon: self.endPoint.lon);
            DispatchQueue.main.async {
                if (r.error == REQUST_ERROR_LOAD_DATA) {
                    Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);

                }
                if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                    Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);

                }
                if (r.error == REQUST_ERROR_INVALID_TOKEN) {
                    Utils.postInvalidTokenNotification();

                }
                if (r.error == REQUST_ERROR_OK) {

                    self.buildRoute(r: r);

                }
            }

        }

    }

    func buildRoute(r: GetDirectionResponce) {
        self.closeRoute(hideRouteView: false);

        let path =  GMSMutablePath(fromEncodedPath: r.points);

        for c in r.meetingPoints {

            var message = "";
            for u in c.users { //todo
                message +=  "\(u.phone)\nWill arrive in \(u.time) seconds\n"
            }

            let marker = MapMarker(position: CLLocationCoordinate2D(latitude: c.latlon.lat, longitude: c.latlon.lon));
            marker?.snippet = message;
            marker?.phone = "MP";
            marker?.isUser = false;
            marker?.icon = MEETING_POINT_EMOJI.image(imageSize: CGSize(width: 15, height: 15), textSize: 12);
            marker?.groundAnchor = CGPoint(x: 0.5, y: 0.5);
            marker?.map = self.map;
            marker?.id = "mp\(marker!.position.latitude)\(marker!.position.longitude)"
            self.routeMarkers.append(marker!);



        }


        self.userRoute = GMSPolyline(path: path);
        self.userRoute?.strokeWidth = 2.0;
        self.userRoute?.strokeColor = Utils.getBackgroundColor();
        self.userRoute?.map = self.map;

        UIView.animate(withDuration: 0.5, animations: {
            self.mapViewYConstraint.constant = 44;
        });
    }

    func locationManager(_ manager: CLLocationManager, didUpdateLocations locations: [CLLocation]) {
        if (self.userMarker == nil) {
            self.userMarker = MapMarker(position: locations[0].coordinate);
```

```
            self.userMarker.title = "Me";
            self.userMarker.phone = CurrentUser.sharedInstance.phone;
            self.userMarker.icon = CurrentUser.sharedInstance.emoji.image(imageSize: CGSize(width: 30, height: 35), textSize: 30);
            self.userMarker.groundAnchor = CGPoint(x: 0.5, y: 0.5);
            self.userMarker.map = self.map;
            self.map.animate(toLocation: locations[0].coordinate);
            self.map.animate(toZoom: 18);
        } else {
            self.userMarker.position = locations[0].coordinate;
            if ((self.map.selectedMarker != nil) && (self.map.selectedMarker == self.userMarker)) {
                self.map.animate(toLocation: self.map.selectedMarker.position);
            }
        }
    }

    override func prepare(for segue: UIStoryboardSegue, sender: Any?) {
        if (segue.identifier == "messages") {
            ((segue.destination as! UINavigationController).viewControllers[0] as! MessagesViewController).groupName = self.groupName;
        }
    }
}

class MapMarker : GMSMarker {

    var isUser = true;
    var id : String = "";
    var updated : Bool = false;
    var phone = "";
}



import UIKit

protocol SelectCountryDelegate {
    func countrySelected(country : Country);
}

class SelectCountryViewController : CustomViewController, UITableViewDelegate, UITableViewDataSource, UISearchResultsUpdating,
UISearchControllerDelegate {

    @IBOutlet weak var tableView: UITableView!;

    let searchController = UISearchController(searchResultsController: nil);

    var countries : [Country] = [];
    var delegate : SelectCountryDelegate!;

    override func viewDidLoad() {
        super.viewDidLoad();

        self.searchController.searchBar.barTintColor = self.tableView.backgroundColor;
        self.searchController.searchBar.returnKeyType = .default;
        self.searchController.view.tintColor = Utils.getMainColor();
        self.searchController.searchBar.keyboardAppearance = .dark;
        (self.searchController.searchBar.value(forKey: "searchField") as? UITextField)?.textColor = Utils.getMainColor();
        self.searchController.searchResultsUpdater = self;
        self.searchController.delegate = self;
        self.searchController.dimsBackgroundDuringPresentation = false;
        self.definesPresentationContext = true;
        self.tableView.tableHeaderView = searchController.searchBar;

        if let path = Bundle.main.path(forResource: "Countries", ofType: "plist") {
            if let countries = NSArray(contentsOfFile: path) {
                for country in countries {
                    let name = (country as AnyObject).object(forKey: "name") as! String;
                    let code = (country as AnyObject).object(forKey: "dial_code") as! String;
                    let flag = (country as AnyObject).object(forKey: "flag") as! String;
                    self.countries.append(Country(name: name, flag: flag, code: code));
                }
            }
        }
```

```
        self.tableView.delegate = self;
        self.tableView.dataSource = self;
        self.tableView.tableFooterView = UIView();
        self.tableView.backgroundView = UIView();

    }

    @IBAction func onClose(_ sender: Any) {
        self.navigationController?.dismiss(animated: true, completion: {

        });
    }



    func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
        return self.countries.count;
    }

    func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {
        let cell = tableView.dequeueReusableCell(withIdentifier: "CountryTableViewCell", for: indexPath) as! CountryTableViewCell;
        cell.initializate(country: self.countries[indexPath.item]);
        return cell;
    }


    func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {
        tableView.cellForRow(at: indexPath)?.isSelected = false;
        self.searchController.isActive = false;

        if let d = self.delegate {
            d.countrySelected(country: self.countries[indexPath.item]);
        }
        self.navigationController?.dismiss(animated: true, completion: {

        });
    }

    func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
        if ((self.countries[indexPath.item].name.lowercased().contains(self.searchController.searchBar.text!.lowercased())) ||
(self.searchController.searchBar.text! == "")) {
            return 42;
        }
        return 0;
    }

    func doSearch() {
        self.tableView.reloadData();
    }

    var searchTimer : Timer? = nil;
    func updateSearchResults(for searchController: UISearchController) {
        if (self.searchTimer != nil) {
            self.searchTimer?.invalidate();
            self.searchTimer = nil;
        }
        self.searchTimer = Timer.scheduledTimer(timeInterval: 0.25, target: self, selector: #selector(SelectCountryViewController.doSearch),
userInfo: nil, repeats: false);

    }

    func willPresentSearchController(_ searchController: UISearchController) {

    }

    func didDismissSearchController(_ searchController: UISearchController) {

    }

}
```

```
struct Country {

   init(name : String, flag : String, code : String) {
      self.name = name;
      self.flag = flag;
      self.code = code;
   }

   var name : String;
   var flag : String;
   var code : String;
}


import UIKit

class MessagesViewController : CustomViewController, UITableViewDelegate, UITableViewDataSource, UITextViewDelegate {

   var groupName : String = "";
   @IBOutlet weak var tableView: UITableView!;
   @IBOutlet weak var text: UITextView!;
   @IBOutlet weak var bottomConstaint: NSLayoutConstraint!;
   @IBOutlet weak var inputHeightConstraint: NSLayoutConstraint!;

   var timer : Timer!;

   lazy var refreshControllTop : UIRefreshControl = {
      let refreshControl = UIRefreshControl();
      refreshControl.addTarget(self, action: #selector(MessagesViewController.updateMessageTop), for: .valueChanged);
      refreshControl.tintColor = .white;
      return refreshControl;
   }();


   @IBOutlet weak var sendButton: UIButton!;

   override func viewDidLoad() {
      super.viewDidLoad();

      NotificationCenter.default.addObserver(self, selector: #selector(MessagesViewController.keyboardWillShow(_:)), name:
NSNotification.Name.UIKeyboardWillShow, object: nil);
      NotificationCenter.default.addObserver(self, selector: #selector(MessagesViewController.keyboardWillHide(_:)), name:
NSNotification.Name.UIKeyboardWillHide, object: nil);

      self.tableView.delegate = self;
      self.tableView.dataSource = self;
      self.tableView.tableFooterView = UIView();
      self.tableView.backgroundView = UIView();

      self.tableView.setContentOffset(CGPoint.init(x: 0, y: CGFloat.greatestFiniteMagnitude), animated: false);

      self.tableView.insertSubview(refreshControllTop, at: 0);

      self.text.delegate = self;
      self.sendButton.isEnabled = false;

      _ = MessageKeeper.sharedInstance(groupName: self.groupName).loadNext();
      self.timer = Timer.scheduledTimer(timeInterval: 2, target: self, selector: #selector(MessagesViewController.timerUpdate), userInfo:
nil, repeats: true);
   }

   var updating : Bool = false;
   func timerUpdate() {
      if (!self.updating) {
         self.updating = true;
         DispatchQueue.global().async {
            let r = MessageKeeper.sharedInstance(groupName: self.groupName).loadNext();

            DispatchQueue.main.async {
               self.updating = false;
               if (r == REQUST_ERROR_LOAD_DATA) {
                  Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
```

```
                }
                if (r == REQUST_ERROR_UNKNOWN_ERROR) {
                    Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                }
                if (r == REQUST_ERROR_INVALID_TOKEN) {
                    Utils.postInvalidTokenNotification();
                }
                if (r == REQUST_ERROR_OK) {
                    self.tableView.reloadData();

                }


            }
        }
    }

    var updatingTop : Bool = false;
    func updateMessageTop() {
        if (!updatingTop) {
            self.updatingTop = true;
            DispatchQueue.global().async {
                let r = MessageKeeper.sharedInstance(groupName: self.groupName).loadPrev();

                DispatchQueue.main.async {
                    self.refreshControllTop.endRefreshing();
                    self.updatingTop = false;
                    if (r == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
                    }
                    if (r == REQUST_ERROR_INVALID_TOKEN) {
                        Utils.postInvalidTokenNotification();
                    }
                    if (r == REQUST_ERROR_OK) {
                        self.tableView.reloadData();

                    }


                }
            }
        }


    }

    @IBAction func onClose(_ sender: Any) {
        self.navigationController?.dismiss(animated: true, completion: {

        });
    }

    @IBAction func onSend(_ sender: Any) {
        let controller = Utils.getWaitViewController();
        self.present(controller, animated: false, completion: nil);

        DispatchQueue.global().async {

            let r = ServerDataLoader.sendMessage(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName, message : self.text.text);
            _ = MessageKeeper.sharedInstance(groupName: self.groupName).loadNext();
            DispatchQueue.main.async {
                controller.dismiss(animated: false, completion: {
                    if (r.error == REQUST_ERROR_LOAD_DATA) {
                        Utils.showAlert(title: "Loading data error", message: "Check your Internet connection", parentViewController: self);
                    }
                    if (r.error == REQUST_ERROR_UNKNOWN_ERROR) {
                        Utils.showAlert(title: "Unknown error", message: "Try again", parentViewController: self);
```

```
            }
            if (r.error == REQUST_ERROR_INVALID_TOKEN) {
               Utils.postInvalidTokenNotification();
            }
            if (r.error == REQUST_ERROR_OK) {
               self.text.text = "";
               self.textViewDidChange(self.text);
               self.view.endEditing(true);
               self.tableView.reloadData();
              //  self.tableView.setContentOffset(CGPoint.init(x: 0, y: CGFloat.greatestFiniteMagnitude), animated: false);
            }
         })



      }
   }


}



func tableView(_ tableView: UITableView, numberOfRowsInSection section: Int) -> Int {
   return MessageKeeper.sharedInstance(groupName: self.groupName).messages.count;
}

func tableView(_ tableView: UITableView, cellForRowAt indexPath: IndexPath) -> UITableViewCell {

   if (MessageKeeper.sharedInstance(groupName: self.groupName).messages[indexPath.item].phone ==
CurrentUser.sharedInstance.phone) {
      let cell = tableView.dequeueReusableCell(withIdentifier: "MessagesTableViewCellRight", for: indexPath) as!
MessagesTableViewCell;
      cell.initializate(message: MessageKeeper.sharedInstance(groupName: self.groupName).messages[indexPath.item].text, avatar:
CurrentUser.sharedInstance.emoji);
      return cell;
   }

   var emoji = MessageKeeper.sharedInstance(groupName: self.groupName).messages[indexPath.item].emoji;
   if (emoji == "") {
      emoji = CURRENT_USER_DEFAULT_EMOJI;
   }

   let cell = tableView.dequeueReusableCell(withIdentifier: "MessagesTableViewCellLeft", for: indexPath) as! MessagesTableViewCell;
   cell.initializate(message: MessageKeeper.sharedInstance(groupName: self.groupName).messages[indexPath.item].text, avatar:
emoji);
   return cell;
}


func tableView(_ tableView: UITableView, didSelectRowAt indexPath: IndexPath) {


}

func tableView(_ tableView: UITableView, heightForRowAt indexPath: IndexPath) -> CGFloat {
   var h = CGFloat(40);
   h += Utils.heightForLabel(text: MessageKeeper.sharedInstance(groupName: self.groupName).messages[indexPath.item].text, font:
UIFont.systemFont(ofSize: 17, weight: UIFontWeightRegular), width: 220);
   return h;
}

func textViewDidChange(_ textView: UITextView) {
   self.sendButton.isEnabled = textView.text.length > 0;

   var h = Utils.heightForLabel(text: textView.text, font: UIFont.systemFont(ofSize: 17, weight: UIFontWeightRegular), width:
textView.bounds.width) + 20; //todo
   if (h < 48) {
      h = 48;
   } else if (h > 200) {
      h  = 200;
```

```
        }
        self.inputHeightConstraint.constant = h;
    }

    func keyboardWillShow(_ notification: Notification) {
        let userInfo = (notification as NSNotification).userInfo!;

        let animationDuration = userInfo[UIKeyboardAnimationDurationUserInfoKey] as! Double;
        let keyboardEndFrame = (userInfo[UIKeyboardFrameEndUserInfoKey] as! NSValue).cgRectValue;

        self.bottomConstaint.constant = keyboardEndFrame.height;
        UIView.animate(withDuration: animationDuration, animations: { () -> Void in
            self.view.layoutIfNeeded();
        })



    }

    func keyboardWillHide(_ notification: Notification) {
        let userInfo = (notification as NSNotification).userInfo!;

        let animationDuration = userInfo[UIKeyboardAnimationDurationUserInfoKey] as! Double;

        self.bottomConstaint.constant = 0.0;
        UIView.animate(withDuration: animationDuration, animations: { () -> Void in
            self.view.layoutIfNeeded();
        })

    }


}

class MessageKeeper {

    private init(groupName : String) {
        self.groupName = groupName;
    }

    private static var shared : MessageKeeper! = nil;
    class func sharedInstance(groupName : String) -> MessageKeeper {
        if ((MessageKeeper.shared != nil) && (MessageKeeper.shared.groupName == groupName)) {
            return MessageKeeper.shared;
        }
        MessageKeeper.shared = MessageKeeper(groupName: groupName);
        _ = MessageKeeper.shared.loadNext();
        return MessageKeeper.shared;
    }

    private var groupName : String;
    var messages : [Message] = [];

    func loadNext() -> Int {
        if (self.messages.count == 0) {
            let r = ServerDataLoader.loadMessages(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName);


            if (r.error == REQUST_ERROR_OK) {

                self.messages.append(contentsOf: r.messages);

            }

            return r.error;
        }
        let r = ServerDataLoader.loadMessagesAfter(phone: CurrentUser.sharedInstance.phone, token: CurrentUser.sharedInstance.token,
groupName: self.groupName,date: messages.last!.date, id: messages.last!.id);


        if (r.error == REQUST_ERROR_OK) {
```

```
            self.messages.append(contentsOf: r.messages);

        }

        return r.error;


    }

    func loadPrev() -> Int {
        if (self.messages.count > 0) {
            let r = ServerDataLoader.loadMessagesBefore(phone: CurrentUser.sharedInstance.phone, token:
CurrentUser.sharedInstance.token, groupName: self.groupName, date: messages[0].date, id: messages[0].id);


            if (r.error == REQUST_ERROR_OK) {
                var m = r.messages;
                m.append(contentsOf: self.messages);
                self.messages = m;
            }

            return r.error;
        }

        return -2;
    }

}


import Foundation


let REQUST_ERROR_LOAD_DATA : Int = -1;
let REQUST_ERROR_OK : Int = 0;
let REQUST_ERROR_INVALID_TOKEN : Int = 1;
let REQUST_ERROR_UNKNOWN_ERROR : Int = -2;
let REQUST_ERROR_INVALID_SMS_CODE : Int = 2;
let REQUST_ERROR_INVALID_GROUP_PASSWORD : Int = 3;
let REQUST_ERROR_ALREADY_CONNECT_GROUP : Int = 4;

let REQUEST_SERVER : String = "https://navinx.appsmithy.com";
//let REQUEST_SERVER : String = "http://10.55.42.191:3000";
let REQUEST_ACT_REGISTRATION : String = "registration";
let REQUEST_ACT_CHECK_CODE : String = "checkCode";
let REQUEST_ACT_SET_EMOJI : String = "setEmoji";
let REQUEST_ACT_GET_GROUPS : String = "getGroups";
let REQUEST_ACT_ADD_GROUP : String = "addGroup";
let REQUEST_ACT_REMOVE_GROUP : String = "removeGroup";

let REQUEST_ACT_ADD_POINT : String = "addPoint";
let REQUEST_ACT_REMOVE_POINT : String = "removePoint";
let REQUEST_ACT_UPDATE_COORDS : String = "updateCoords";
let REQUEST_ACT_GET_GROUP : String = "getGroup";
let REQUEST_ACT_GET_DIRECTION : String = "getDirection";
let REQUEST_ACT_GET_MESSAGES : String = "getMessages";
let REQUEST_ACT_SEND_MESSAGE : String = "postMessage";

class Responce {

    init(d : Data) {
        if String(data: d, encoding: .utf8) != nil {
            print("_____");
            print(String(data: d, encoding: .utf8)!);
            print("_____");
            let json = JSON(data : d);
            if let error = json["error"].int {
                self.error = error;
                self.json = json;
            } else {
                self.error = REQUST_ERROR_LOAD_DATA;
```

```
            }
        } else {
            self.error = REQUST_ERROR_INVALID_TOKEN;
        }



    }

    init() {
        self.error = REQUST_ERROR_LOAD_DATA;
    }

    var error : Int;
    var json : JSON!;

}

class CheckCodeResponce : Responce {



    override init(d : Data) {
        super.init(d: d);
        if (self.error == REQUST_ERROR_OK) {
            self.token = json["token"].string!;
            self.emoji = json["emoji"].string!.replacingOccurrences(of: "\"", with: "");
        }

    }


    override init() {
        super.init();
    }

    var token : String = "";
    var emoji : String = "";

}

class GetGroupsResponce : Responce {

    override init(d : Data) {
        super.init(d: d);
        if (self.error == REQUST_ERROR_OK) {
            for e in json["groups"].array! {
                self.groups.append(e["name"].string!);
            }

        }

    }

    override init() {
        super.init();
    }

    var groups : [String] = [];

}

struct GroupUser {

    init(phone : String, lat : Double, lon : Double, emoji : String) {
        self.phone = phone;
        self.lat = lat;
        self.lon = lon;
        self.emoji = emoji != "" ? emoji : CURRENT_USER_DEFAULT_EMOJI;
        self.name = self.phone;
    }
```

```
    var phone : String;
    var name : String;
    var lat : Double;
    var lon : Double;
    var emoji : String;


}

struct GroupPoint {

    init(name : String, lat : Double, lon : Double) {
        self.name = name;
        self.lat = lat;
        self.lon = lon;
    }

    var name : String;
    var lat : Double;
    var lon : Double;


}

class GetGroupResponce : Responce {

    override init(d : Data) {
        super.init(d: d);
        if (self.error == REQUST_ERROR_OK) {
            for u in json["users"].array! {
                let phone = u["phone"].string!;
                let emoji = u["emoji"].string!.replacingOccurrences(of: "\"", with: "");
                if let lat = u["lat"].double {
                    if let lon = u["lon"].double {
                        self.users.append(GroupUser(phone: phone, lat: lat, lon: lon, emoji: emoji));
                    }

                } else {
                    self.offlineUsers.append(GroupUser(phone: phone, lat: 0, lon: 0, emoji: emoji));
                }

            }

            for p in json["points"].array! {
                let name = p["name"].string!;
                let lat = p["lat"].double!;
                let lon = p["lon"].double!;
                self.points.append(GroupPoint(name: name, lat: lat, lon: lon));
            }
        }

    }

    override init() {
        super.init();
    }

    var users : [GroupUser] = [];
    var offlineUsers : [GroupUser] = [];
    var points : [GroupPoint] = [];

}

struct MeetingUser {

    init(phone : String, time : Int) {
        self.phone = phone;
        self.time = time;
    }

    var time : Int;
    var phone : String;
```

```
}

struct MeetingPoint {

    init(latlon : LatLon, users : [MeetingUser]) {
        self.latlon = latlon;
        self.users = users;
    }

    var users : [MeetingUser];
    var latlon : LatLon;
}

struct LatLon {

    init(lat : Double, lon : Double) {
        self.lat = lat;
        self.lon = lon;
    }

    var lat : Double;
    var lon : Double;
}

class GetDirectionResponce : Responce {

    override init(d : Data) {
        super.init(d: d);
        if (self.error == REQUST_ERROR_OK) {
            for p in json["meetings"].array! {
                let lat = p["lat"].double!;
                let lon = p["lon"].double!;

                var users : [MeetingUser] = [];

                for u in p["users"].array! {
                    let phone = u["phone"].string!;
                    let time = u["time"].int!;
                    users.append(MeetingUser(phone: phone, time: time));
                }


                self.meetingPoints.append(MeetingPoint(latlon: LatLon(lat: lat, lon: lon), users: users));
            }
            self.points = json["points"].string!;
        }

    }

    override init() {
        super.init();
    }

    var points : String = "";
    var meetingPoints : [MeetingPoint] = [];
}

struct Message {

    init(phone : String, groupName : String, text : String, date : String, id : String, emoji : String) {
        self.phone = phone;
        self.groupName = groupName;
        self.text = text;
        self.date = date;
        self.id = id;
        self.emoji = emoji;
    }

    var phone : String;
    var groupName : String;
    var text : String;
    var date : String;
```

```
    var id : String;
    var emoji : String;
}

class GetMessagesResponce : Responce {

    override init(d : Data) {
        super.init(d: d);
        if (self.error == REQUST_ERROR_OK) {
            for m in json["messages"].array! {

                let phone = m["phone"].string!;
                let emoji = m["emoji"].string!;
                let groupName = m["groupName"].string!;
                let text = m["text"].string!;
                let date = m["date"].string!;
                let id = m["id"].string!;

                self.messages.append(Message(phone: phone, groupName: groupName, text: text, date: date, id: id, emoji: emoji));
            }
        }

    }

    override init() {
        super.init();
    }

    var messages : [Message] = [];
}

class ServerDataLoader {


    private class func loadData(url : URL, data : String) -> Data! {
        let session = URLSession.shared;
        var request = URLRequest(url: url);
        request.httpMethod = "POST";
        request.httpBody = data.data(using: .utf8);
        request.addValue("application/json", forHTTPHeaderField: "Content-Type");
        request.addValue("application/json", forHTTPHeaderField: "Accept");
        let res = session.synchronousDataTask(with: request);
        return res;
    }

    private class func buildURL(act : String) -> URL {
        let stringURL = REQUEST_SERVER + "/" + act;
        return URL(string: stringURL)!;
    }

    class func registration(phone : String) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_REGISTRATION);
        let responce = ServerDataLoader.loadData(url: requestURL, data:data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();

    }

    class func checkCode(phone : String, code : String) -> CheckCodeResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"code\" : \"\(code)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_CHECK_CODE);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return CheckCodeResponce(d: responce!);
        }

        return CheckCodeResponce();
```

```
    }


    class func setEmoji(phone : String, token : String, emoji : String) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"emoji\" : \"\(emoji)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_SET_EMOJI);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

    class func getGroups(phone : String, token : String) -> GetGroupsResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_GROUPS);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return GetGroupsResponce(d: responce!);
        }
        return GetGroupsResponce();
    }

    class func addGroup(phone : String, token : String, groupName : String, groupPassword : String) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"groupPassword\" :
\"\(groupPassword)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_ADD_GROUP);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

    class func addPoint(phone : String, token : String, groupName : String, pointName : String, lat : Double, lon : Double) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"pointName\" :
\"\(pointName)\", \"lat\" : \(lat), \"lon\" : \(lon)}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_ADD_POINT);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

    class func removePoint(phone : String, token : String, groupName : String, pointName : String, lat : Double, lon : Double) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"pointName\" :
\"\(pointName)\", \"lat\" : \(lat), \"lon\" : \(lon)}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_REMOVE_POINT);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

    class func removeGroup(phone : String, token : String, groupName : String) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_REMOVE_GROUP);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }
```

```
    class func updateCoords(phone : String, token : String, groupName : String, lat : Double, lon : Double) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"lat\" : \(lat), \"lon\" :
\(lon)}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_UPDATE_COORDS);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

    class func getGroup(phone : String, token : String, groupName : String) -> GetGroupResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_GROUP);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);

        if (responce != nil) {
            return GetGroupResponce(d: responce!);
        }
        return GetGroupResponce();
    }

    class func getDirection(phone : String, token : String, groupName : String, startLat : Double, startLon : Double, endLat : Double, endLon :
Double) -> GetDirectionResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"startLat\" :
\(startLat), \"startLon\" : \(startLon), \"endLat\" : \(endLat), \"endLon\" : \(endLon)}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_DIRECTION);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);
        if (responce != nil) {
            return GetDirectionResponce(d: responce!);
        }
        return GetDirectionResponce();
    }

    class func loadMessagesBefore(phone : String, token : String, groupName : String, date : String, id : String) -> GetMessagesResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"params\" : {
\"messageId\" : \"\(id)\", \"date\" : \"\(date)\", \"direction\" : \"before\" }}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_MESSAGES);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);
        if (responce != nil) {
            return GetMessagesResponce(d: responce!);
        }
        return GetMessagesResponce();
    }

    class func loadMessagesAfter(phone : String, token : String, groupName : String, date : String, id : String) -> GetMessagesResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"params\" : {
\"messageId\" : \"\(id)\", \"date\" : \"\(date)\", \"direction\" : \"after\" }}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_MESSAGES);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);
        if (responce != nil) {
            return GetMessagesResponce(d: responce!);
        }
        return GetMessagesResponce();
    }

    class func loadMessages(phone : String, token : String, groupName : String) -> GetMessagesResponce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_GET_MESSAGES);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);
        if (responce != nil) {
            return GetMessagesResponce(d: responce!);
        }
        return GetMessagesResponce();
    }

    class func sendMessage(phone : String, token : String, groupName : String, message : String) -> Responce {
        let data : String = "{ \"phone\" : \"\(phone)\", \"token\" : \"\(token)\", \"groupName\" : \"\(groupName)\", \"text\" :
\"\(message.replacingOccurrences(of: "\n", with: "\\n"))\"}";
        let requestURL : URL = buildURL(act: REQUEST_ACT_SEND_MESSAGE);
        let responce = ServerDataLoader.loadData(url: requestURL, data: data);
```

```
        if (responce != nil) {
            return Responce(d: responce!);
        }
        return Responce();
    }

}


import UIKit

class GroupTableViewCell : UITableViewCell {
    @IBOutlet weak var groupName: UILabel!

    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(false, animated: animated)
    }

    func initializate(groupName : String) {
        self.groupName.text = groupName;

        let bgColorView = UIView();
        bgColorView.backgroundColor = UIColor(red: 0.2, green: 0.2, blue: 0.2, alpha: 1.0);
        self.selectedBackgroundView = bgColorView;
    }

}



import UIKit

class CountryTableViewCell : UITableViewCell {

    @IBOutlet weak var name: UILabel!;
    @IBOutlet weak var code: UILabel!;


    override func awakeFromNib() {
        super.awakeFromNib()
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(false, animated: animated)
    }

    func initializate(country : Country) {
        self.name.text = "\(country.flag) \(country.name)";
        self.code.text = "+\(country.code)";

        let bgColorView = UIView();
        bgColorView.backgroundColor = UIColor(red: 0.2, green: 0.2, blue: 0.2, alpha: 1.0);
        self.selectedBackgroundView = bgColorView;
    }

}




import UIKit

class MessagesTableViewCell : UITableViewCell {

    @IBOutlet weak var width: NSLayoutConstraint!;

    @IBOutlet weak var avatar: UILabel!;
```

```
    @IBOutlet weak var message: UILabel!;


    override func awakeFromNib() {
        super.awakeFromNib();
    }

    override func setSelected(_ selected: Bool, animated: Bool) {
        super.setSelected(false, animated: animated);
    }

    func initializate(message : String, avatar : String) {
        self.avatar.text = avatar;
        self.message.text = message;

        var w = Utils.widthForLabel(text: message, font: self.message.font) + 16;

        if (w > 236) {
            w = 236;
        }
        self.width.constant = w;


    }


}


import Foundation
import Alamofire
import AlamofireNetworkActivityLogger

class AlamofireRequest: RequestManager {


    internal func request<T: Decodable, E: Decodable & NetworkError>(
        requestModel: Encodable?,
        url: URL,
        type: RequestHTTPMethod,
        headers: [String : String]?,
        completion: @escaping (_ model: T?, _ backendError: E?) -> Void)  {

        self.request(requestModel: requestModel, url: url, type: type, headers: headers) { (responce, error, _) in
            completion(responce, error)
        }
    }

    internal func request<T: Decodable, E: Decodable & NetworkError>(
        requestModel: Encodable?,
        url: URL,
        type: RequestHTTPMethod,
        headers: [String : String]?,
        completion: @escaping (_ model: T?, _ backendError: E?, _ statusCode: Int?) -> Void
        ) {

        Alamofire.request(url, method: type.alamofireHTTPMethod, parameters: requestModel?.dictionary, encoding: JSONEncoding.default,
headers: headers).responseData { (response) in
            if (response.result.isSuccess) {
                if ((response.response?.statusCode ?? 0) / 100 == 2) { //200, 201 ... 299
                    if let v = response.result.value {
                        completion(try? JSONDecoder().decode(T.self, from: v), nil, response.response?.statusCode)
                    }
                } else {
                    if let v = response.result.value {
                        let error: E? = try? JSONDecoder().decode(E.self, from: v)
                        error?.statusCode = response.response?.statusCode
                        completion(nil, error, response.response?.statusCode)
                    }
                }
            } else {
                completion(nil, nil, response.response?.statusCode)
```

```
        }
      }

   }

   internal func request(
      url: URL,
      type: RequestHTTPMethod,
      headers: [String : String]?,
      parameters: [String : String]?,
      completion: @escaping (_ result: String?, _ statusCode: Int?) -> Void
      ) {

      Alamofire.request(url, method: type.alamofireHTTPMethod, parameters: parameters, encoding: JSONEncoding.default, headers:
headers).responseData { (response) in
         if let data = response.result.value {
            let result = String(data: data, encoding: .utf8)
            completion(result, response.response?.statusCode)
         } else {
            completion(nil, response.response?.statusCode)
         }

      }
   }

   required init() {
      #if DEBUG
      NetworkActivityLogger.shared.level = .debug
      NetworkActivityLogger.shared.startLogging()
      #endif
   }



}

extension Encodable {
   var dictionary: [String: Any]? {
      guard let data = try? JSONEncoder().encode(self) else { return nil }
      return (try? JSONSerialization.jsonObject(with: data, options: .allowFragments)).flatMap { $0 as? [String: Any] }
   }
}

extension RequestHTTPMethod {
   var alamofireHTTPMethod: Alamofire.HTTPMethod {
      switch self {
      case .get:
         return .get
      case .post:
         return .post
      case .put:
         return .put
      case .patch:
         return .patch
      case .delete:
         return .delete
      }
   }
}


import Foundation

class RequestHelper {

   static let shared: RequestHelper = RequestHelper()

   private(set) var requestManager: RequestManager?

   private init() {

   }
```

```
    func setupRequestManaget(requestManager: RequestManager) {
        self.requestManager = requestManager
    }

}

protocol Request {

}

import Foundation

enum RequestHTTPMethod: String {
    case get = "GET"
    case post = "POST"
    case put = "PUT"
    case patch = "PATCH"
    case delete = "DELETE"
}

protocol RequestManager {
    init()
    func request<T: Decodable, E: Decodable & NetworkError>(
        requestModel: Encodable?,
        url: URL,
        type: RequestHTTPMethod,
        headers: [String : String]?,
        completion: @escaping (_ model: T?, _ backendError: E?, _ statusCode: Int?) -> Void
    )

    func request<T: Decodable, E: Decodable & NetworkError>(
        requestModel: Encodable?,
        url: URL,
        type: RequestHTTPMethod,
        headers: [String : String]?,
        completion: @escaping (_ model: T?, _ backendError: E?) -> Void
    )

    func request(
        url: URL,
        type: RequestHTTPMethod,
        headers: [String : String]?,
        parameters: [String : String]?,
        completion: @escaping (_ result: String?, _ statusCode: Int?) -> Void    )
}
import Foundation


enum NavinxoURL: String {
    #if DEV
    case BASE_URL = ""
    case ODM_DATA_URL = "https://api.openstreetmap.org/api/0.6/map?bbox=30.232,50.212,30.244,50.627"
    #else
    case BASE_URL = ""
    case ODM_DATA_URL = "https://api.openstreetmap.org/api/0.6/map?bbox=30.232,50.212,30.244,50.627"

    #endif


}

protocol Endpoint {
    var rawValue: String {get}
}

enum NavinxoEndpoint: String, Endpoint {
    case test = ""
}

class URLBuilder {
    class func build(endpoint: Endpoint) -> URL {
```

```
    return URLBuilder.build(endpoint: endpoint.rawValue)
  }

  class func build(url: String = NavinxoURL.BASE_URL.rawValue, endpoint: String) -> URL {
    let urlString = ((url + endpoint).addingPercentEncoding(withAllowedCharacters: CharacterSet.urlQueryAllowed) ?? "")
    return URL(string: urlString)!
  }
}



import Foundation

class NetworkError: Error {
  var statusCode: Int?

  init() {

  }
}


        import Foundation


        public class AStar {
          private var totalVertices: Set<Vertex>


          public init(vertices: Set<Vertex>) {
            totalVertices = vertices
          }


          private func clearCache() {
            totalVertices.forEach { $0.clearCache() }
          }


          public func findShortestPaths(from startVertex: Vertex) {
            clearCache()
            var currentVertices = self.totalVertices
            startVertex.pathLengthFromStart = 0
            startVertex.pathVerticesFromStart.append(startVertex)
            var currentVertex: Vertex? = startVertex
            while let vertex = currentVertex {
              currentVertices.remove(vertex)
              let filteredNeighbors = vertex.neighbors.filter { currentVertices.contains($0.0) }
              for neighbor in filteredNeighbors {
                let neighborVertex = neighbor.0
                let weight = neighbor.1


                let theoreticNewWeight = vertex.pathLengthFromStart + weight
                if theoreticNewWeight < neighborVertex.pathLengthFromStart {
                  neighborVertex.pathLengthFromStart = theoreticNewWeight
                  neighborVertex.pathVerticesFromStart = vertex.pathVerticesFromStart
                  neighborVertex.pathVerticesFromStart.append(neighborVertex)
                }
              }
              if currentVertices.isEmpty {
                currentVertex = nil
                break
              }
              currentVertex = currentVertices.min { $0.pathLengthFromStart < $1.pathLengthFromStart }
            }
          }
        }
```